UNIVERSIDAD DE LA RIOJA

TRABAJO FIN DE ESTUDIOS

Título

# STAFFit: Plataforma web modular para la gestión integral de recursos humanos


Autor/es

IGNACIO MARTÍNEZ PÉREZ

Director/es

ELOY JAVIER MATA SOTÉS

Facultad

Facultad de Ciencia y Tecnología


Titulación

Grado en Ingeniería Informática


Departamento

MATEMÁTICAS Y COMPUTACIÓN


Curso académico

2019-20





publicaciones.unirioja.es
E-mail: publicaciones@unirioja.es

UNIVERSIDAD DE LA RIOJA

**Facultad de Ciencia y Tecnología**

# TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

STAFFit: Plataforma web modular para la gestión integral de recursos humanos


Realizado por:

Ignacio Martínez Pérez

Tutelado por:

Eloy Javier Mata Sotés


**Logroño, junio, 2020**

## RESUMEN

El presente Trabajo de Fin de Grado nació como propuesta de las empresas Bosonit S.L. y Nter Solutions S.L. con dos objetivos principales: unificar los sistemas y procesos que han de llevar a cabo sus trabajadores en una sola plataforma web ampliable en el futuro, y poder ofrecer esta plataforma de manera comercial a otras empresas y entidades.

Esta plataforma recibe el nombre de STAFFit, y para cumplir con los dos objetivos anteriores se apoya en un diseño modular. Este diseño nos permite desarrollar módulos independientes que posteriormente se pueden integrar a STAFFit. También nos permite ofrecer una plataforma a medida al cliente creando módulos personalizados.

Tras un gran cambio en la dirección y alcance del proyecto propuesto inicialmente por la empresa a finales de abril, este proyecto ya no se ajustaba al producto exigido por la empresa. Dada la inviabilidad que suponía empezar el proyecto desde cero por lo avanzado que estaba su desarrollo y la proximidad de las fechas de entrega, se decidió continuar con el desarrollo de la propuesta inicial de manera independiente a la empresa.

El resultado final de este trabajo es la creación de la plataforma web STAFFit tal cual se propuso inicialmente por la empresa. Esta plataforma cuenta con tres módulos: el módulo de recursos humanos, el módulo de empleados y el módulo de criptomonedas.



## ABSTRACT

The present Bachelor's Degree Final Project was born as a proposal of the companies *Bosonit S.L.* and *Nter Solutions S.L.* with two main objectives: to unify the systems and processes to be carried out by the staff in a single web platform expandable in the future and to be able to offer this platform commercially to other companies and entities.

This web platform is called STAFFit, and to fulfill the two previous objectives it is based on a modular design. This design allows us to develop independent modules which subsequently can be integrated into STAFFit. It also allows us to offer a platform tailored to the client by creating custom modules.

After a major change in the course and scope of the project initially proposed by the company at the end of April, this project no longer met the product required by the company. Given the unfeasibility of starting the project from scratch due to the advanced stage of development and the proximity of the deadline, it was decided to continue with the development of the initial proposal independently from the company.

The result of this project is the creation of the STAFFit web platform as initially proposed by the company. This platform has three modules: the human resources module, the employee module and the cryptocurrency module.

# ÍNDICE

# 1. INTRODUCCIÓN

## 1.1. Antecedentes

Bosonit S.L. y Nter S.L. son dos compañías españolas especializadas en análisis de datos y Big Data y en desarrollo software empresarial respectivamente. Ambas compañías están ampliamente vinculadas, compartiendo dirección y departamento de recursos humanos. A su vez forman parte del Grupo Nfq, un grupo de consultoría independiente formado, junto con las anteriores, por otras dos empresas: Nfq Advisory y Nforce.

Ambas empresas han experimentado un rápido y fuerte crecimiento, superando los 180 empleados de manera conjunta en menos de 5 años. Esto se ha conseguido en gran parte gracias a su fuerte apuesta en el talento joven y en su formación, acogiendo a un gran número de estudiantes en prácticas.

Hasta ahora el departamento de recursos humanos ha estado utilizando varios sistemas diferentes para llevar a cabo las tareas de gestión, lo que provocaba problemas de comunicación y productividad que se fueron agravando a medida que aumentaba la plantilla. Era necesario buscar una alternativa más eficaz.

Es así como nació la propuesta de crear una plataforma web que unificase todos los procesos de gestión de recursos humanos en una única herramienta, sustituyendo así los sistemas empleados anteriormente. A esta plataforma se le ha dado el nombre de STAFFit.

Con el tiempo STAFFit fue creciendo en funcionalidades, por lo que ya no se limita a ser una herramienta de gestión de recursos humanos. También es un portal para los empleados desde el cual pueden llevar a cabo diferentes tareas e incorpora un sistema de incentivos para los empleados basado en blockchain mediante el uso de una criptomoneda propia.

Para poder implementar todas estas funcionalidades y permitir integrar nuevas características o incluso otros sistemas directamente en el futuro, se ha decidido dotar a STAFFit de un diseño modular.

El diseño modular es el pilar de STAFFit, y nos permite cumplir con varios de los objetivos marcados para este proyecto. Estos objetivos se explican detalladamente en el siguiente punto.

## 1.2. Objetivos

El objetivo inicial STAFFit era sustituir las diferentes herramientas utilizadas por el departamento de recursos humanos por una solo herramienta desde la que se pudieran hacer todas las tareas de gestión.

Más tarde se decidió que STAFFit no solo unificara los procesos de gestión de recursos humanos, sino que también unificará los procesos de todos los trabajadores y fuera un portal de empleados.

Viendo las grandes posibilidades que podía tener STAFFit como servicio a ofrecer a otras compañías de manera comercial, se añadió a los dos objetivos anteriores la capacidad de que STAFFit fuera una plataforma fácilmente personalizable y que permitiera integrar otros sistemas existentes.

Para cumplir con estos objetivos nos ayuda el hecho de que STAFFit tenga un diseño modular. Este diseño también permite que los módulos pueden ser desarrollados por equipos de trabajo independientes, agilizando el desarrollo.

Podemos resumir los objetivos anteriores en:

- Sustituir los diferentes sistemas empleados por los trabajadores y los procesos que han de realizar por una única plataforma web.
- Crear una plataforma ampliable, ya sea creando nuevos módulos o integrando sistemas ya existentes.
- Permitir crear una plataforma a medida para el cliente con módulos personalizados.

# 2. PLANIFICACIÓN

La planificación es una de las partes más importantes y delicadas de un proyecto. Así como una buena planificación puede ayudarnos a que nuestro proyecto se desarrolle de manera satisfactoria en sus distintas fases, una mala planificación puede provocar que surjan complicaciones que lleven a no conseguir el resultado esperado o incluso a que el proyecto finalmente fracase. Es por esto por lo que hay que prestar especial dedicación a esta fase.

En esta sección se abordan las diferentes partes de las que consta la planificación de este proyecto.

## 2.1. Actores

Los actores que se están involucrados en este proyecto son:

- Eloy Javier Mata Sotés. Tutor del TFG y profesor del Departamento de Matemáticas y Computación.
- María Díez Arnaiz. Mánager del departamento de Recursos Humanos de Bosonit y Nter.
- Miguel Fernández Morales. Director general de Bosonit y Nter.
- Ignacio Martínez Pérez. Creador del TFG y estudiante en el Grado en Ingeniería Informática.

## 2.2. Tecnologías

A continuación, se explican brevemente las tecnologías utilizadas en STAFFit, distinguiendo entre las tecnologías utilizadas en el back-end, en el front-end y las relacionadas con Blockchain.

### Back-end

*PostgreSQL*

La base de datos que se emplea en STAFFit está creada con PostgreSQL, un sistema de gestión de bases de datos de código abierto.

*GraphQL*

La API utilizada en STAFFit está construida en GraphQL, un lenguaje de consulta de datos para APIs. GraphQL es una creación de Facebook, que buscaba una alternativa a las APIs REST tradicionales. Actualmente es un software de código abierto.

*Hasura GraphQL engine*

La API GraphQL que utiliza STAFFit está creada con Hasura GraphQL engine. Hasura es un engine que proporciona una API en GraphQL para bases de datos hechas con Postgres. Además, ofrece una interfaz gráfica desde la que podemos visualizar, crear y modificar las tablas.

*Auth0*

La autenticación y la gestión de los permisos de los usuarios dentro de STAFFit se lleva a cabo con Auth0, una plataforma para gestionar la identidad.

*Funciones serverless*

La lógica de negocio de STAFFit se realiza mediante eventos creados con Hasura. Estos eventos se ejecutan en forma de funciones serverless. Una función serverless permite ejecutar un fragmento de código en un proveedor en la nube en vez de ejecutarlo en el servidor que estamos utilizando para nuestro back-end. Como desarrolladores, esto nos facilita el trabajo, ya que nos abstrae de la infraestructura subyacente. Además, conseguimos quitar carga de trabajo a nuestro servidor, que solo se encargará de realizar las operaciones relacionadas con nuestra base de datos.

En STAFFit utilizamos el servicio en la nube Glitch para crear funciones serverless. Cada una de estas funciones consisten en pequeñas aplicaciones Node.js construidas sobre Express.js, el framework web más popular de Node.js.

## Front-end

*JavaScript y React*

Para la construcción de las interfaces de usuario utilizamos React, una librería de JavaScript. Los motivos por los que se ha elegido React y no otro framework de JavaScript para desarrollo de interfaces como AngularJS o Vue.js es por su mayor simplicidad y popularidad, que se traduce en un mejor soporte de la comunidad.

*Apollo*

Para conectarnos a la API GraphQL que nos proporciona Hasura utilizamos Apollo. Apollo es una plataforma que permite a nuestra aplicación conectarse a múltiples APIs, bases de datos y microservicios, gestionando todas estas conexiones en un único lugar.

*Material UI*

Para el diseño de las interfaces de STAFFit se ha utilizado principalmente Material UI, una librería que nos ayuda a implementar un diseño simple y elegante en las interfaces que hemos construido con React de manera sencilla.

## Blockchain

*Ethereum*

La plataforma blockchain sobre la que se va a desarrollar la criptomoneda que se usará en STAFFit es Ethereum, ya que es la plataforma con mayor soporte y número de herramientas disponibles para crear soluciones blockchain.

*Solidity*

La criptomoneda se va a desarrollar con Solidity, un lenguaje de programación creado por Ethereum específicamente para el desarrollo de smart contracts.

*Truffle*

Truffle es el marco de desarrollo más popular para trabajar con Ethereum. Facilita la creación de proyectos blockchain en todas sus etapas: desarrollo, testeo y despliegue del proyecto.

## 2.3. Metodología

STAFFit es un proyecto que va a ir incrementando sus funcionalidades durante todo su desarrollo y el cliente ha solicitado disponer de una versión funcional lo antes posible. Además, no hay una definición clara del proyecto, pudiéndose producir cambios en cualquier fase. Por tanto, la metodología que mejor se adecúa a estos dos requisitos es una metodología iterativo-incremental. Cada una de las iteraciones sigue el siguiente flujo:

1. **Análisis**. Se analiza qué es lo que el cliente quiere y se definen los requisitos que tiene el sistema o funcionalidad a implementar.
2. **Diseño**. Se determina cómo va a implementarse el sistema o funcionalidad.
3. **Implementación**. Se construye e integra el sistema o funcionalidad.
4. **Testeo**. Se evalúa el resultado obtenido, haciendo las pruebas necesarias para verificar el correcto funcionamiento del sistema o funcionalidad implementada.

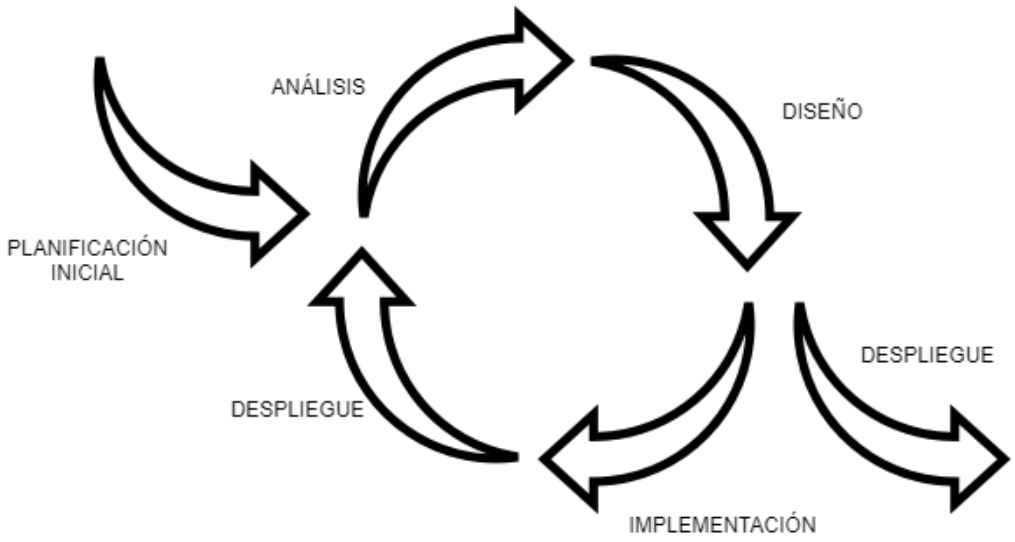


*Figura 1. Modelo iterativo-incremental*

## 2.4. Estructura de descomposición de trabajo (EDT)

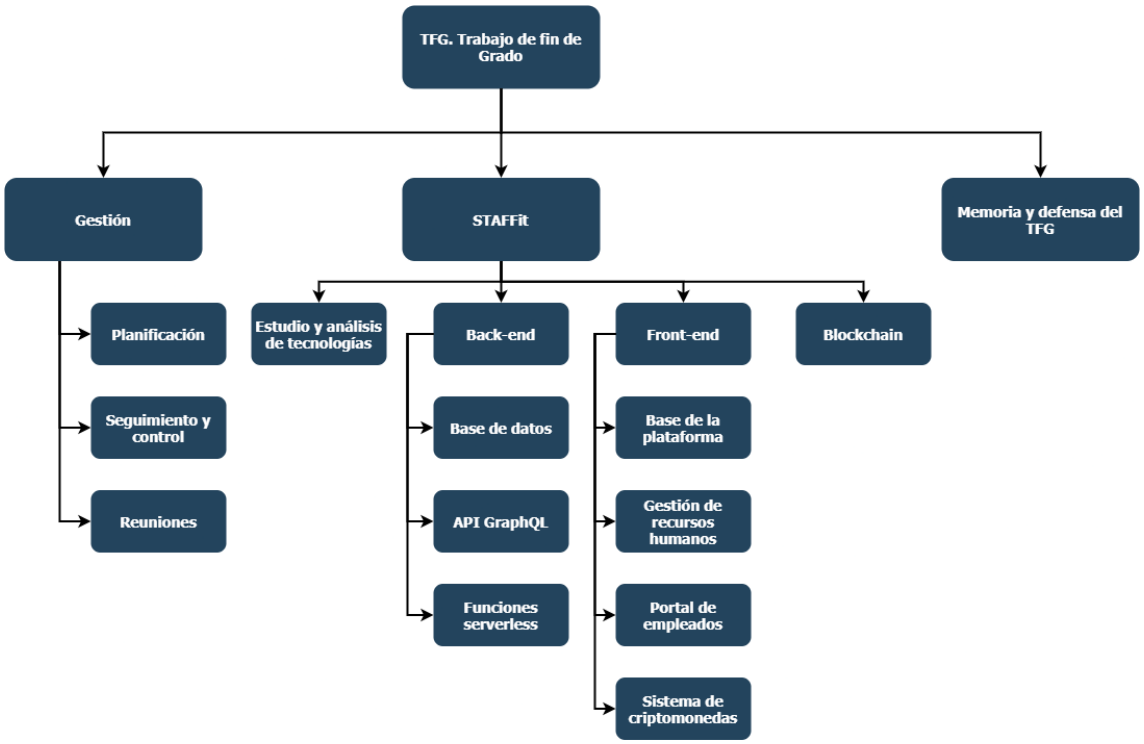


*Figura 2. Estructura de descomposición de tareas*

## 2.5. Diccionario de la EDT

| ID | NOMBRE | DESCRIPCIÓN |
|---|---|---|
| **1** | **Gestión del proyecto** | |
| 1.1 | Planificación | Planificación de las tareas a realizar en el proyecto y estimación de tiempo y recursos que se van a necesitar. |
| 1.2 | Seguimiento y control | Supervisión de la ejecución del proyecto para identificar los problemas y adoptar las medidas adecuadas. |
| 1.3 | Reuniones | Reuniones que se van a llevar a cabo con el cliente y con el tutor académico. |
| **2** | **Desarrollo del proyecto** | |
| 2.1 | Estudio y análisis de tecnologías | Búsqueda de las tecnologías más adecuadas para nuestro proyecto y formación necesaria para emplearlas. |
| 2.2 | Back-end | Creación de la base de datos, de la API para comunicarse con la misma y de los eventos mediante funciones serverless. |
| *2.2.1* | *Base de datos* | Diseño e implementación de la base de datos utilizando PostgreSQL. |
| *2.2.2* | *API GraphQL* | Creación de la API GraphQL mediante la herramienta Hasura GraphQL Engine |
| *2.2.3* | *Funciones serverless* | Creación de funciones serverless en el servicio Glitch para la ejecución de eventos desde la API GraphQL. |
| 2.3 | Front-end | Creación de la interfaz de la aplicación web en React. |

| | | |
|---|---|---|
| *2.3.1* | *Base de la plataforma* | Diseño e implementación de la pantalla de inicio de sesión y de los elementos de la interfaz comunes a los tres módulos. |
| *2.3.2* | *Gestión de recursos humanos* | Diseño e implementación de la interfaz del módulo de gestión de recursos humanos. |
| *2.3.3* | *Portal de empleados* | Diseño e implementación de la interfaz del módulo de empleados. |
| *2.3.4* | *Sistema de criptomonedas* | Diseño e implementación de la interfaz del módulo de empleados. |
| 2.4 | Blockchain | Desarrollo de los smart contracts de la criptomoneda y despliegue en la blockchain de Ethereum. |
| **3** | **Memoria y defensa del TFG** | |
| 3.1 | Memoria | Realización de la memoria del TFG. |
| 3.2 | Presentación | Preparación de la de defensa del TFG. |

*Tabla 1. Diccionario de la EDT*

## 2.6. Estimación de tiempos

| PAQUETES DE TRABAJO | | TIEMPO ESTIMADO |
|---|---|---|
| **ID** | **NOMBRE** | |
| **1** | **Gestión del proyecto** | **40 h** |
| 1.1 | Planificación | 15 h |
| 1.2 | Seguimiento y control | 15 h |
| 1.3 | Reuniones | 10 h |
| **2** | **Desarrollo del proyecto** | **270 h** |
| 2.1 | Estudio y análisis de tecnologías | 25 h |
| 2.2 | Back-end | 25 h |
| *2.2.1* | *Base de datos* | 15 h |
| *2.2.2* | *API GraphQL* | 5 h |
| *2.2.3* | *Funciones serverless* | 5 h |
| 2.3 | Front-end | 150 h |
| *2.3.1* | *Base de la plataforma* | 20 h |
| *2.3.2* | *Gestión de recursos humanos* | 60 h |
| *2.3.3* | *Portal de empleados* | 30 h |
| *2.3.4* | *Criptomonedas* | 40 h |
| 2.4 | Blockchain | 70 h |
| **3** | **Memoria y defensa del TFG** | **35 h** |
| 3.1 | Memoria | 25 h |
| 3.2 | Presentación | 10 h |
| **TOTAL** | | **345 h** |

*Tabla 2. Estimación de tiempos*

## 2.7. Diagrama de Gantt

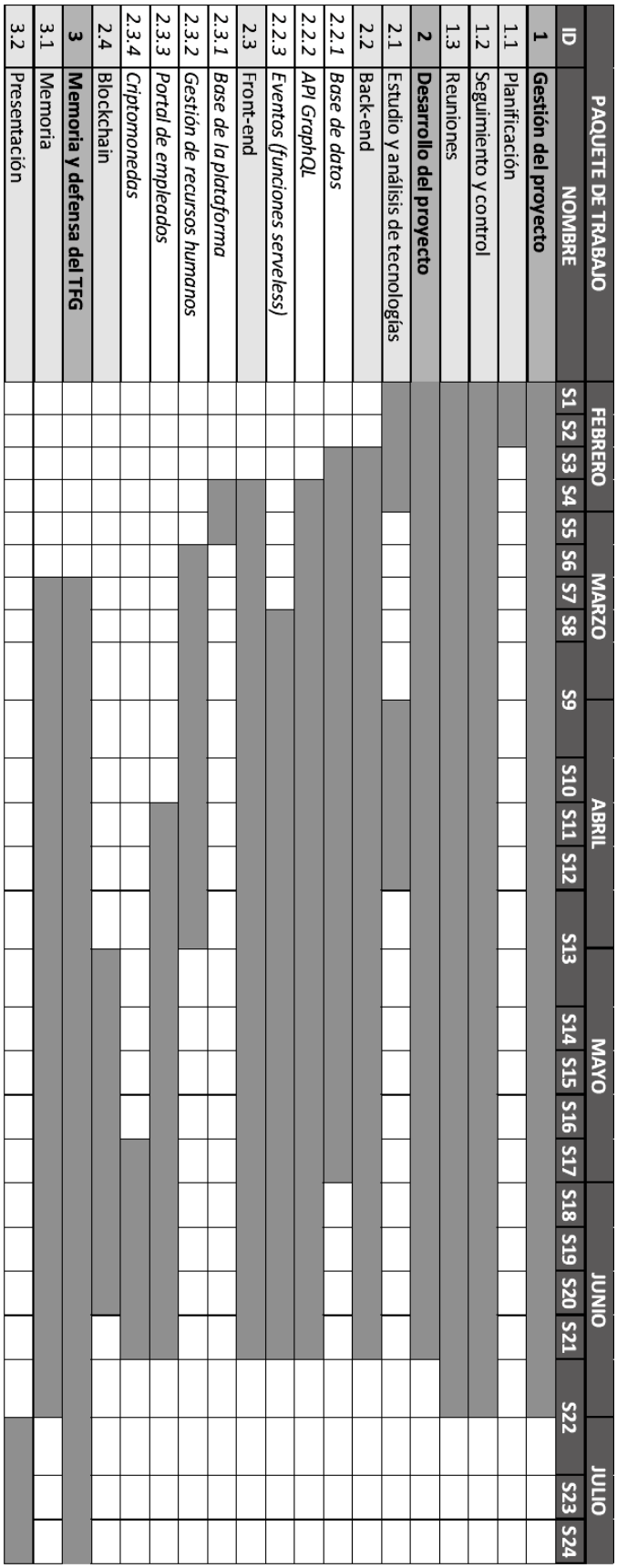


*Tabla 3. Diagrama de Gantt*

## 2.8. Diagrama de hitos

**Febrero**

| Lu | Ma | Mi | Ju | Vi | Sá | Do |
|---|---|---|---|---|---|---|
| | | | | | 1 | 2 |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 |
| 24 | 25 | 26 | 27 | 28 | 29 | |

**Marzo**

| Lu | Ma | Mi | Ju | Vi | Sá | Do |
|---|---|---|---|---|---|---|
| | | | | | | 1 |
| 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| 16 | 17 | 18 | 19 | 20 | 21 | 22 |
| 23 | 24 | 25 | 26 | 27 | 28 | 29 |
| 30 | 31 | | | | | |

**Abril**

| Lu | Ma | Mi | Ju | Vi | Sá | Do |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | | | |

**Mayo**

| Lu | Ma | Mi | Ju | Vi | Sá | Do |
|---|---|---|---|---|---|---|
| | | | | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 |

**Junio**

| Lu | Ma | Mi | Ju | Vi | Sá | Do |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |
| 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| 29 | 30 | | | | | |

**Julio**

| Lu | Ma | Mi | Ju | Vi | Sá | Do |
|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 |
| 27 | 28 | 29 | 30 | 31 | | |

- 3 de febrero: Inicio del TFG
- 13 de marzo: Finalización del módulo de gestión de recursos humanos
- 3 de abril: Finalización del módulo de empleados
- 12 de junio: Finalización del módulo de criptomonedas
- 22 de junio: Depósito de la memoria del TFG
- 13 de julio: Defensa del TFG

## 2.9. Gestión de riesgos

| PLAN DE RIESGOS | | | |
|---|---|---|---|
| FUENTE | RIESGO | ESTRATEGIA DE PREVENCIÓN/MINIMIZACIÓN | PLAN DE CONTINGENCIA |
| Proyecto | Falta de tiempo | Realizar una buena planificación. Centrarse primero en los requisitos más importantes. Hacer un seguimiento del trabajo para evitar imprevistos. | Acordar con el cliente un cambio en el alcance del proyecto. Acordar con el cliente que requisitos tienen mayor prioridad. |
| | Fallos de seguridad o pérdida de datos | Realizar un análisis previo de las posibles brechas de seguridad y hacer un diseño acorde a este análisis. Realizar copias de seguridad periódicas. | Comunicar al cliente el problema en cuanto se tenga conocimiento de el mismo para tomar las medidas oportunas. Restringir el acceso a la aplicación hasta que se haya solucionado el problema. |
| Cliente | Mala comunicación | Centrarse en las cosas aclaradas mientras se reestablece la comunicación. | Aclarar todo lo posible lo antes posible. |
| | Retrasos en el feedback | Centrarse en las cosas aclaradas mientras se reestablece la comunicación. | Aclarar todo lo posible lo antes posible para que el feedback necesario sea el mínimo. |
| | Cambios en las especificaciones | Recortar el alcance lo posible y centrarse en ajustar lo que ya tenemos. | Explicar bien el impacto de ciertos cambios. |
| | Cambios en el alcance | Repartir las tareas, aunque se disminuya la calidad. | Explicar bien el impacto de ciertos cambios. |
| Tecnología | Tecnología no adecuada | Realizar un buen análisis de las tecnologías más convenientes y estudiar posibles alternativas. | Usar una tecnología alternativa. |
| | Poca experiencia con la tecnología a utilizar | Realizar un estudio previo de las herramientas. | Usar una tecnología alternativa o buscar a alguien experimentado que nos asista. |

*Tabla 4. Plan de riesgos*

# 3. ANÁLISIS

## 3.1. Roles de usuario

Nuestra aplicación va a contar con tres roles diferentes para controlar el acceso que tiene el usuario tanto a nivel de interfaz como a nivel de base de datos. Estos tres roles se han definido previamente con el cliente, y son los siguientes:

- Empleado. Será el rol de usuario por defecto. Su nivel de acceso será el más limitado.
- Mánager. Este rol de usuario será asignado a los mánagers de cada CAR[1]. Tendrán ciertos permisos adicionales a los usuarios con rol de empleado.
- Administrador. Es el rol de usuario con mayor nivel de acceso. Será asignado a los empleados del departamento de recursos humanos.

## 3.2. Captura de requisitos

Requisitos funcionales

| Requisitos de sesión de usuario | |
|---|---|
| Código | Descripción |
| RFS-01 | El sistema permitirá que el usuario acceda a la aplicación con su correo corporativo. |
| RFS-02 | El sistema permitirá al usuario salir de la aplicación borrando su sesión. |

*Tabla 5.1. Requisitos de sesión de usuario*

| Requisitos del módulo de recursos humanos | |
|---|---|
| Código | Descripción |
| RFH-01 | El módulo contará con los siguientes apartados: solicitudes, campañas, gestión de empleados e informe. |
| RFH-02 | El sistema permitirá al usuario crear solicitudes de personal introduciendo los siguientes datos: nombre, descripción, número de personas, fecha de incorporación, tecnologías, estado, compañía, CAR y categoría. Solo podrán realizar esta operación los usuarios con rol de administrador o mánager. |
| RFH-03 | El sistema permitirá al usuario editar las solicitudes de personal que ha creado. Solo podrán realizar esta operación los usuarios con rol de administrador o mánager. |
| RFH-04 | El sistema permitirá al usuario visualizar todas las solicitudes registradas. Solo podrán realizar esta operación los usuarios con rol de administrador o mánager. |
| RFH-05 | El sistema contará con un muro de comentarios para cada solicitud de personal en el que el usuario podrá escribir observaciones sobre la solicitud en cuestión. Solo será accesible a los usuarios con rol de administrador o mánager. |
| RFH-06 | El sistema permitirá a los administradores marcar una solicitud de personal como importante. |

[1] CAR son las siglas de *Célula de Alto Rendimiento*, y es como se denominan a los grupos de trabajo en las empresas Bosonit S.L. y Nter S.L.

| | |
|---|---|
| RFH-07 | El sistema permitirá a los administradores crear campañas de incorporación de personal introduciendo los siguientes datos: origen, descripción, ciudad, estado, responsable, plazas garantizadas, plazas no garantizadas, fecha de incorporación y fecha de finalización. |
| RFH-08 | El sistema permitirá a los administradores editar las campañas registradas. |
| RFH-09 | El sistema permitirá a los administradores visualizar todas las campañas registradas. |
| RFH-10 | El sistema permitirá al usuario crear nuevas solicitudes de personal asociadas a una campaña de incorporación introduciendo los siguientes datos: nombre, plazas pedidas, compañía y CAR. Solo podrán realizar esta operación los usuarios con rol de administrador o mánager. |
| RFH-11 | El sistema permitirá a los administradores dar de alta nuevos empleados introduciendo los siguientes datos: nombre, apellidos, correo, teléfono, fecha de incorporación, compañía, CAR, categoría, ciudad, contrato, jornada, remuneración, estado y tecnologías. |
| RFH-12 | El sistema enviará un correo automatizado a recursos humanos cada vez que se dé de alta un nuevo empleado. |
| RFH-13 | El sistema permitirá a los administradores modificar los datos de las altas de nuevos empleados. |
| RFH-14 | El sistema permitirá a los administradores eliminar los empleados dados de alta. |
| RFH-15 | El sistema permitirá a los administradores visualizar un informe realizado con PowerBI sobre las solicitudes de personal, campañas de incorporación y altas de nuevos empleados registrados en el sistema. |

*Tabla 5.2. Requisitos del módulo de recursos humanos*

| Requisitos del módulo de empleado | |
|---|---|
| Código | Descripción |
| RFE-01 | El sistema permitirá al usuario visualizar sus datos personales. |
| RFE-02 | El sistema permitirá al usuario ver su información corporativa. |
| RFE-03 | El sistema permitirá enviar notificaciones al usuario. |

*Tabla 5.3. Requisitos del módulo de empleado*

| Requisitos del módulo de criptomonedas | |
|---|---|
| Código | Descripción |
| RFC-01 | El sistema permitirá al usuario crear una cartera de criptomonedas si aún no dispone de una. |
| RFC-02 | El sistema permitirá al usuario visualizar su cartera de criptomonedas. |
| RFC-03 | El sistema permitirá al usuario transferir sus criptomonedas a otro usuario. |
| RFC-04 | El sistema permitirá al usuario canjear sus criptomonedas por productos y servicios ofrecidos por la empresa. |
| RFC-05 | El sistema permitirá al usuario visualizar su historial de transacciones. |
| RFC-06 | El sistema permitirá a los administradores crear criptomonedas. |
| RFC-07 | El sistema permitirá a los administradores repartir criptomonedas entre los empleados. |
| RFC-08 | El sistema permitirá a los administradores ver el número total de criptomonedas en circulación. |

| | |
|---|---|
| RFC-09 | El sistema permitirá a los administradores visualizar el historial de todas las transacciones realizadas por los empleados. |

*Tabla 5.4. Requisitos del módulo de criptomonedas*

## Requisitos no funcionales

| Requisitos técnicos | |
|---|---|
| Código | Descripción |
| RNF-01 | La aplicación estará adaptada para su uso desde ordenador. |
| RNF-02 | La aplicación deberá utilizar el sistema de autenticación de Google para permitir el acceso con el correo corporativo sin necesidad de registro. |
| RNF-03 | La aplicación estará alojada en un servidor en la nube. |
| RNF-04 | La base de datos deberá estar alojada en un servidor en la nube. |

*Tabla 5.5. Requisitos técnicos*

# 4. DISEÑO

## 4.1. Arquitectura

A continuación, se explica brevemente la arquitectura que emplea STAFFit. Las tecnologías que se nombrarán a continuación están explicadas de manera más detallada en el apartado *2.2. Tecnologías*.

El front-end del sistema se ha construido con React, una librería de Javascript para el desarrollo de interfaces web. Desde el front-end nos conectaremos tanto al back-end de la aplicación como a la blockchain para hacer uso de la criptomoneda.

La criptomoneda que se emplea en la aplicación se ha construido sobre Ethereum. No está desplegada sobre la red principal de Ethereum, sino que está desplegada en una red de pruebas pública llamada Ropsten. Esto se debe a que la criptomoneda está en fase experimental y no se quiere gastar dinero real para llevar a cabo transacciones.

El back-end cuenta con tres partes diferenciadas: el sistema de autenticación de usuario, la API y la lógica de negocio. Para el sistema de autenticación se ha utilizado la plataforma Auth0, que además también controla el rol que tiene el usuario dentro de la aplicación. La lógica de negocio no se lleva a cabo dentro de la API como es habitual, sino que se realiza a parte mediante el uso de funciones serverless que se ejecutan en el servicio en la nube Glitch. La API que utiliza la aplicación está construida en GraphQL utilizando la herramienta Hasura GraphQL Engine, y es el eje central del back-end. Además de comunicarse con la base de datos, la API se comunica con el sistema de autenticación para controlar a qué información de la base de datos tiene acceso el usuario dependiendo del rol que tiene asignado dentro de la aplicación. También se comunica con Glitch cada vez que es necesario hacer lógica de negocio.

La base de datos está creada con PostgreSQL y está alojada en el servicio en la nube Heroku.

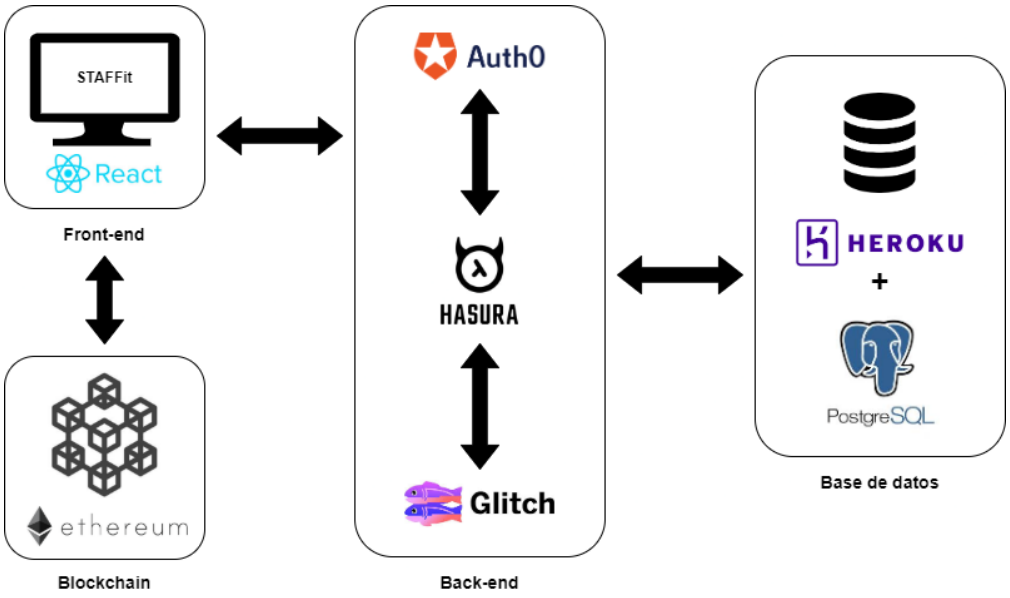


*Figura 3. Arquitectura de STAFFit*

## 4.2. Diseño de la base de datos

### Descripción

Como ya hemos descrito anteriormente, STAFFit cuenta con tres módulos: el módulo de empleados, el módulo de gestión de recursos humanos y el módulo de criptomonedas.

Para el módulo de empleado es necesario almacenar los datos personales de los empleados, así como su información corporativa (fecha de incorporación, empresa en la que trabaja, CAR al que pertenece, etc.).

El módulo de gestión de recursos humanos es el que más uso de la base de datos va a hacer. Tendremos que almacenar las campañas de incorporación de personal, las solicitudes de personal y las incorporaciones de empleados. La información de estas tres áreas está interrelacionada.

Por último, para el módulo de criptomonedas es necesario almacenar la información relativa a las carteras de los empleados y a las transacciones que han hecho dentro de la aplicación.

Además de todos estos datos también se hace uso de varias tablas auxiliares que son usadas por más de un módulo. Por ejemplo, tablas como los CAR o las tecnologías que se usan dentro de la empresa son usadas tanto por el módulo de empleado como por el módulo de recursos humanos.

Debido al gran número de tablas y de campos que hay en nuestra base de datos solo nos limitaremos a mostrar la parte del diseño de la base de datos con las tablas y campos más relevantes para entender su funcionamiento y las relaciones entre las tablas.

Diagrama entidad-relación

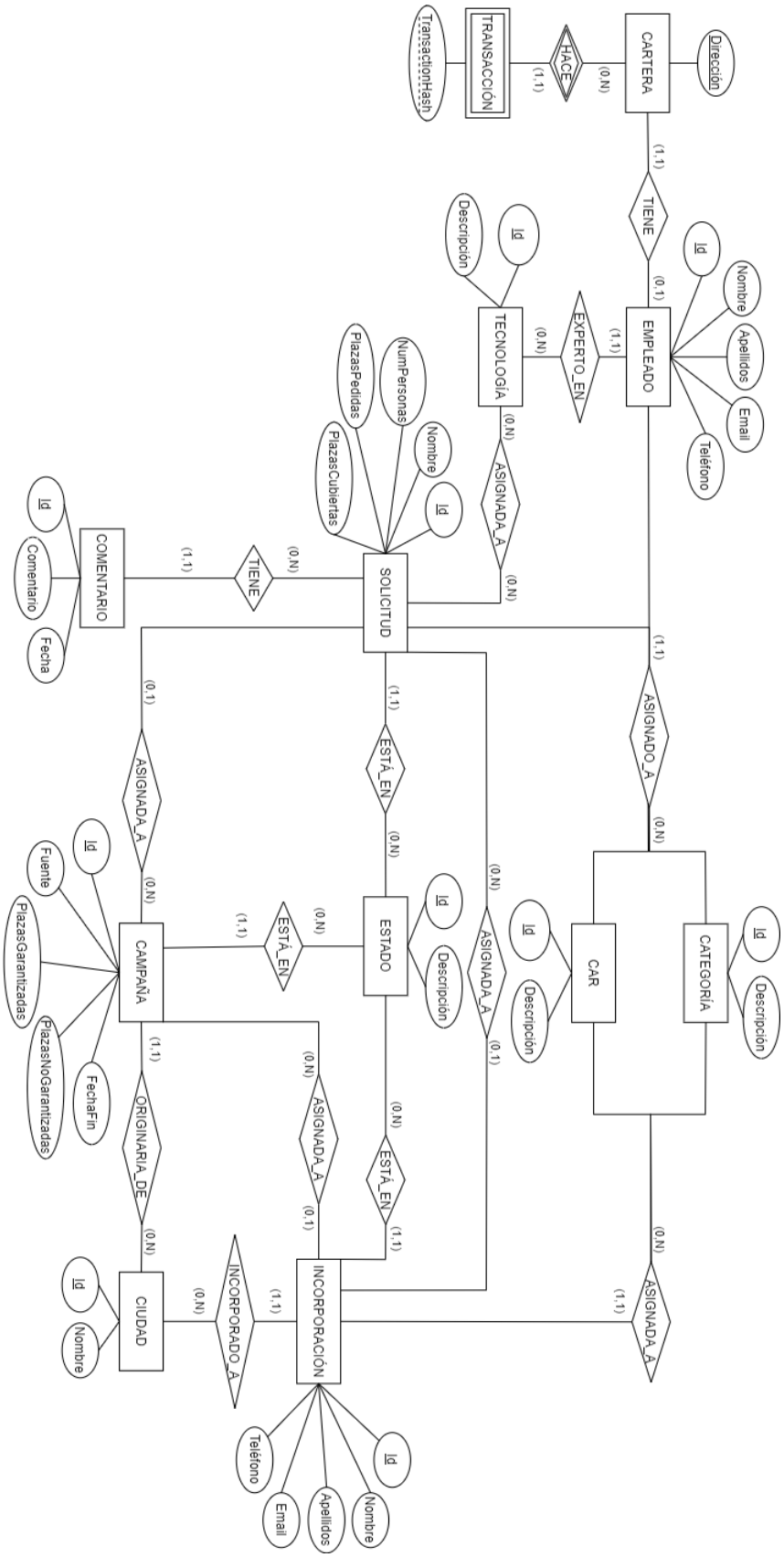


*Figura 4. Diagrama entidad-relación*

## Modelo relacional

EMPLEADO

| Id | Nombre | Apellidos | Email | Teléfono | Tecnología | Categoría | CAR |
|---|---|---|---|---|---|---|---|
| | | | | | CE: Tecnología | CE: Categoría | CE: CAR |

CARTERA

| Dirección | Empleado |
|---|---|
| | CE: Empleado |

TRANSACCIÓN

| Cartera | TransactionHash |
|---|---|
| CE: Cartera | |

TECNOLOGÍA

| Id | Descripción |
|---|---|

CATEGORÍA

| Id | Descripción |
|---|---|

CAR

| Id | Descripción |
|---|---|

SOLICITUD

| Id | Nombre | NumPersonas | PlazasPedidas | PlazasCubiertas | Estado | Campaña |
|---|---|---|---|---|---|---|
| | | | | | CE: Estado | CE: Campaña |

COMENTARIO

| Id | Comentario | Fecha | Solicitud |
|---|---|---|---|
| | | | CE: Solicitud |

ESTADO

| Id | Descripción |
|---|---|

CAMPAÑA

| Id | Fuente | PlazasGarantizadas | PlazasNoGarantizadas | FechaFin | Estado | Ciudad |
|---|---|---|---|---|---|---|
| | | | | | CE: Estado | CE: Ciudad |

INCORPORACIÓN

| Id | Nombre | Apellidos | Email | Teléfono | Ciudad | Campaña | Solicitud |
|---|---|---|---|---|---|---|---|
| | | | | | CE: Ciudad | CE: Campaña | CE: Solicitud |

| Estado | Categoría | CAR |
|---|---|---|
| CE: Estado | CE: Categoría | CE: CAR |

CIUDAD

| Id | Nombre |
|---|---|

SOLICITUD_TECNOLOGÍA

| Solicitud | Tecnología |
|---|---|
| CE: Solicitud | CE: Tecnología |

## 4.3. Diseño de la interfaz

El diseño de la interfaz se ha realizado en consenso con el cliente. A continuación, se muestra el resultado de la interfaz propuesta en forma de prototipos.

### Inicio de sesión

La interfaz de inicio de sesión de nuestra aplicación es sencilla. Constará de una imagen corporativa y en lado izquierdo dispondrá de un enlace para acceder con la cuenta corporativa sin necesidad de registro previo. Para ello se usará la autenticación de Google tal como ha exigido el cliente.

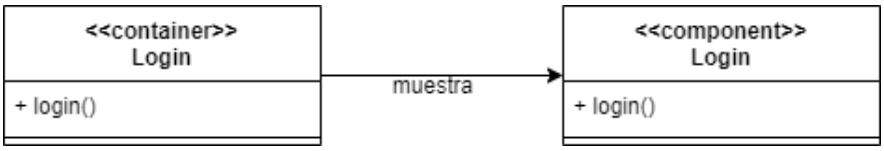


*Figura 5.1.1. Estructura lógica pantalla de inicio de sesión*

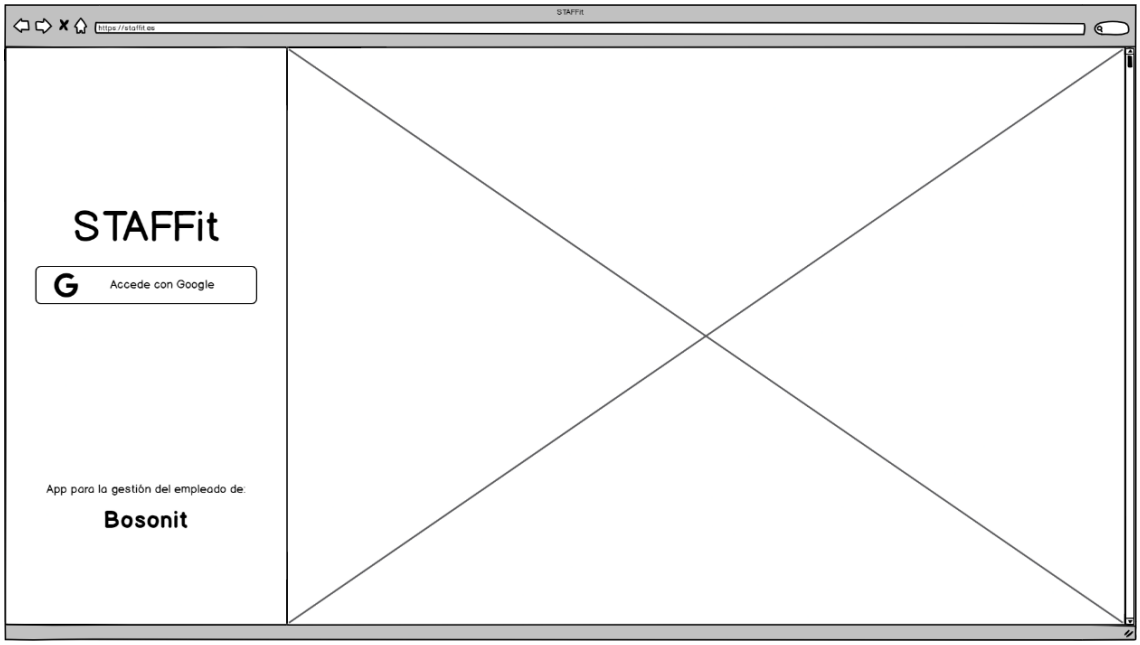


*Figura 5.1.2. Prototipo pantalla de inicio de sesión*

### Módulo de empleado

La barra de navegación y el menú lateral serán comunes en todas las pantallas. Las opciones disponibles en la barra de navegación de izquierda a derecha serán las siguientes:

ocultar/mostrar el menú lateral, enlace a tus notificaciones, enlace a tu perfil y cerrar sesión. En el menú lateral podremos ver tres apartados que corresponden a los tres módulos de nuestra aplicación: *My STAFFit* (módulo de empleado), *Recruitment* (módulo de recursos humanos) y *Bosocoin* (módulo de criptomonedas).

El módulo de empleado contará con una sola pantalla, y será la primera pantalla que verá el usuario una vez que haya iniciado sesión. Dispondrá de tres secciones: *Mi perfil*, *Notificaciones* y *Mi cartera*. En *Mi perfil* podrás visualizar tus datos personales y tus datos relativos a la empresa. La sección *Notificaciones* mostrará una visión general de tus alertas y te permitirá descartarlas. Por último, en *Mi cartera* se mostrará una gráfica con la evolución temporal de tu balance de criptomonedas, así como tu balance actual.

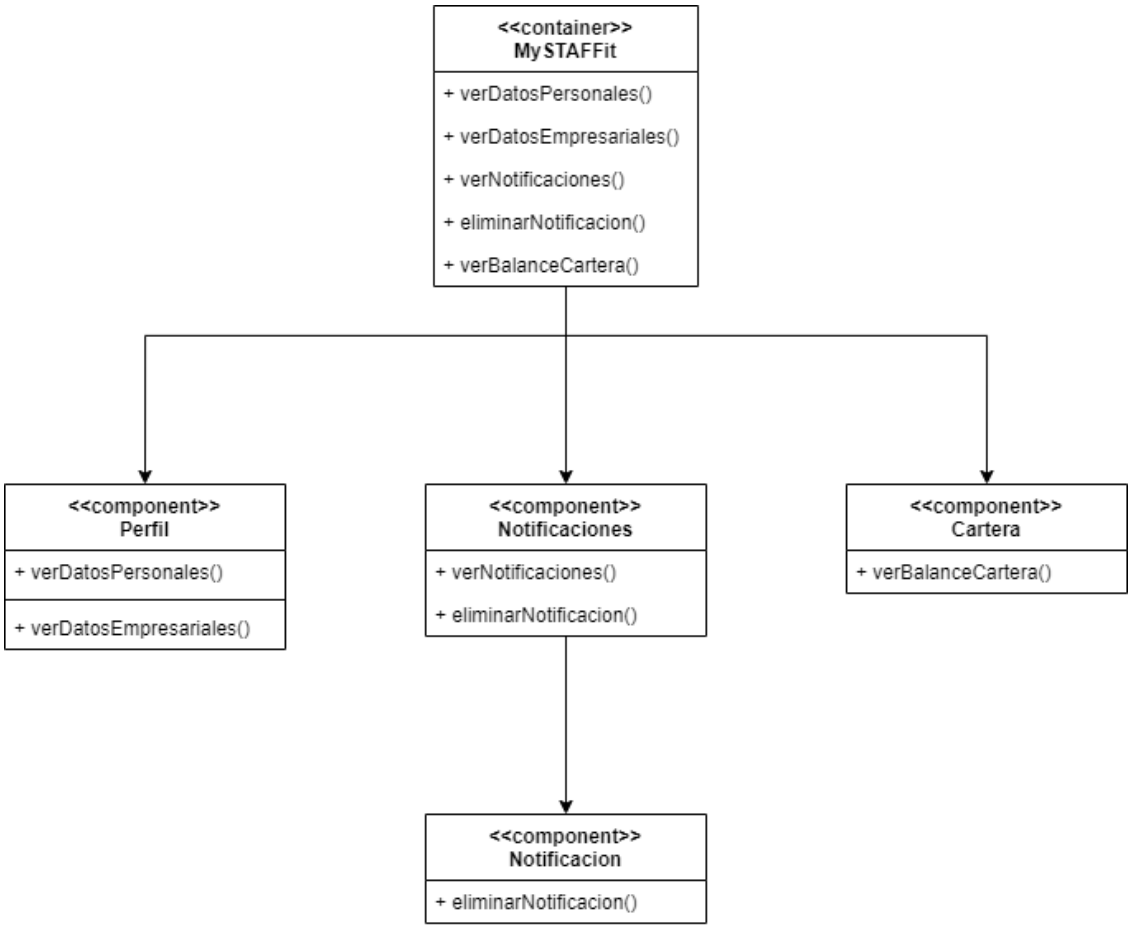


*Figura 5.2.1. Estructura lógica pantalla My STAFFit*

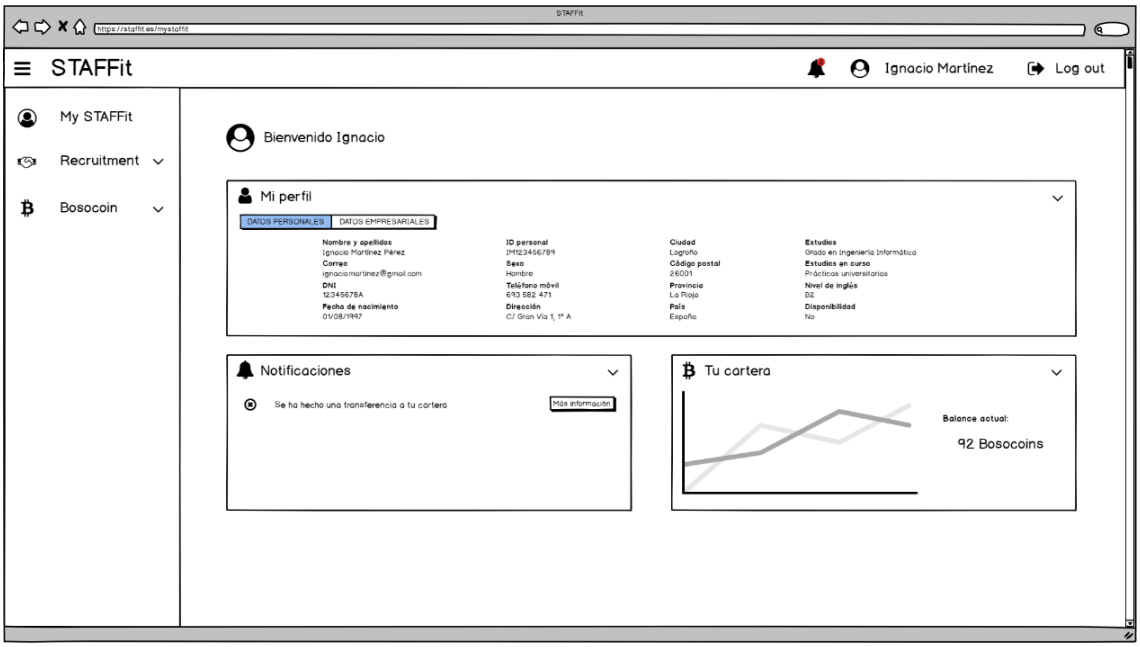


*Figura 5.2.2. Prototipo pantalla My STAFFit*

## Módulo gestión de recursos humanos

El módulo de gestión de recursos humanos solo será accesible por los empleados con rol de mánager o administrador. Contará con cuatro apartados: *Solicitudes*, *Campañas*, *Informe* y *Gestión de empleados*.

El apartado *Solicitudes* será accesible tanto por mánagers como administradores y dispondrá de dos secciones. En la primera sección se mostrarán las campañas de incorporación abiertas en forma de carrusel, y cada una contará con un enlace en la parte inferior que permitirá acceder a un formulario para crear una solicitud de personal asociada a dicha campaña. En la segunda sección se mostrará un enlace que permitirá acceder a un formulario para crear solicitudes que no estén asociadas a ninguna campaña y una tabla con todas las solicitudes. La tabla dispondrá de un buscador de solicitudes por nombre y también permitirá filtrar las solicitudes de la tabla por los campos empresa, CAR y estado de la solicitud. Cada solicitud de la tabla contará con tres botones: uno para editar la petición, otro para acceder a la sección de comentarios de la solicitud y otro para marcar la solicitud como crítica.

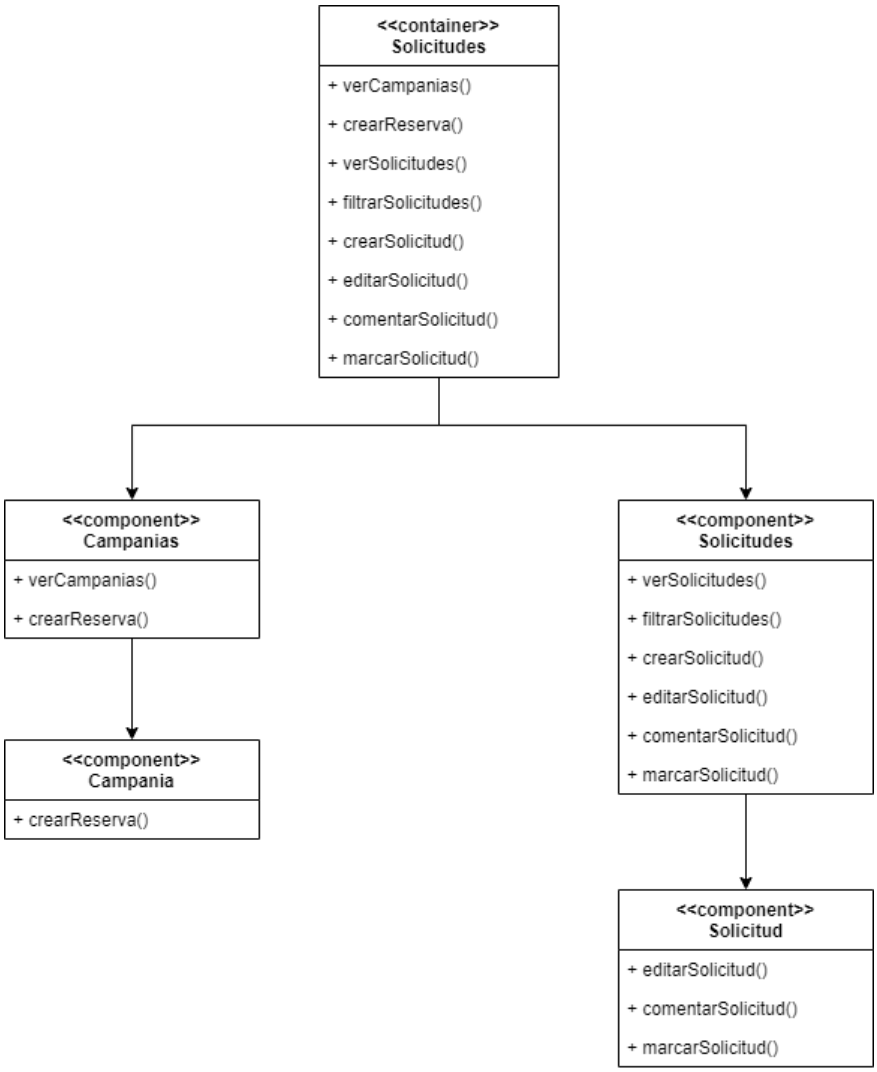


*Figura 5.3.1. Estructura lógica pantalla de solicitudes*

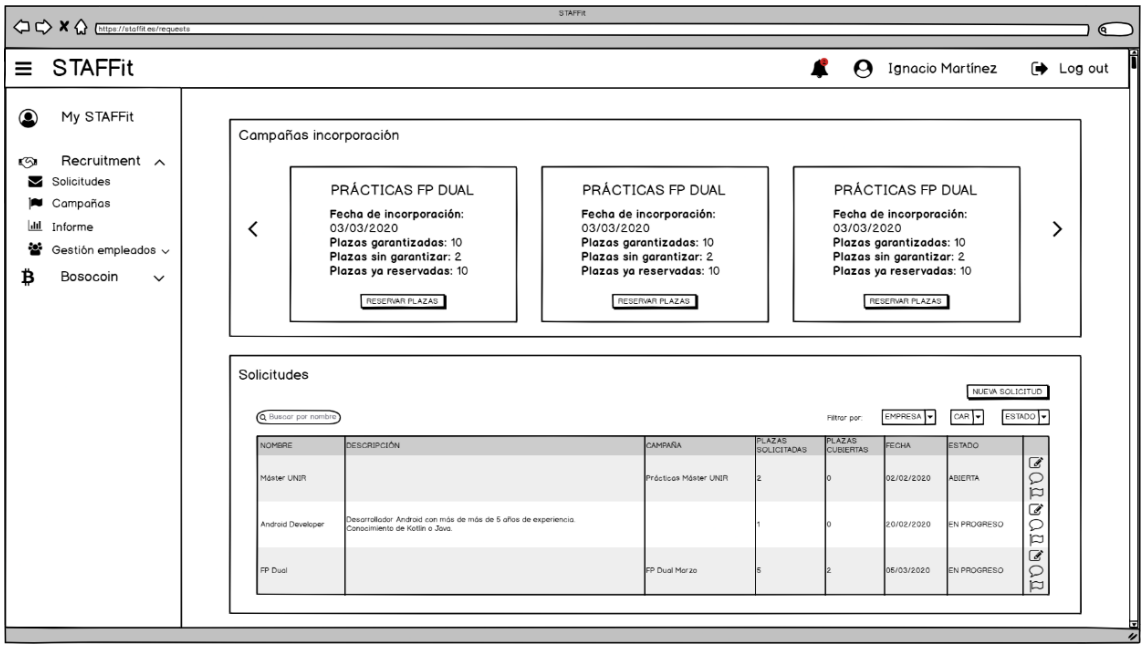


*Figura 5.3.2. Prototipo pantalla de solicitudes*

El apartado *Campañas* solo será accesible por los administradores. Dispondrá de un slider en la parte superior para seleccionar el mes y debajo mostrará una tabla con todas las campañas creadas en el mes seleccionado. Cada campaña de la tabla contará con un enlace para acceder al formulario de edición de la campaña. También dispondrá de un enlace que permitirá acceder a un formulario para crear una nueva campaña de incorporación.

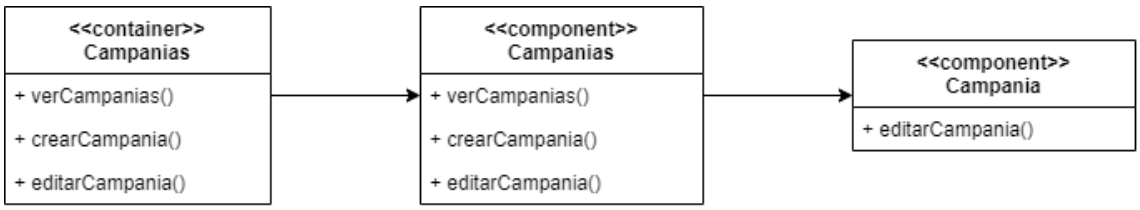


*Figura 5.4.1. Estructura lógica pantalla de campañas de incorporación*

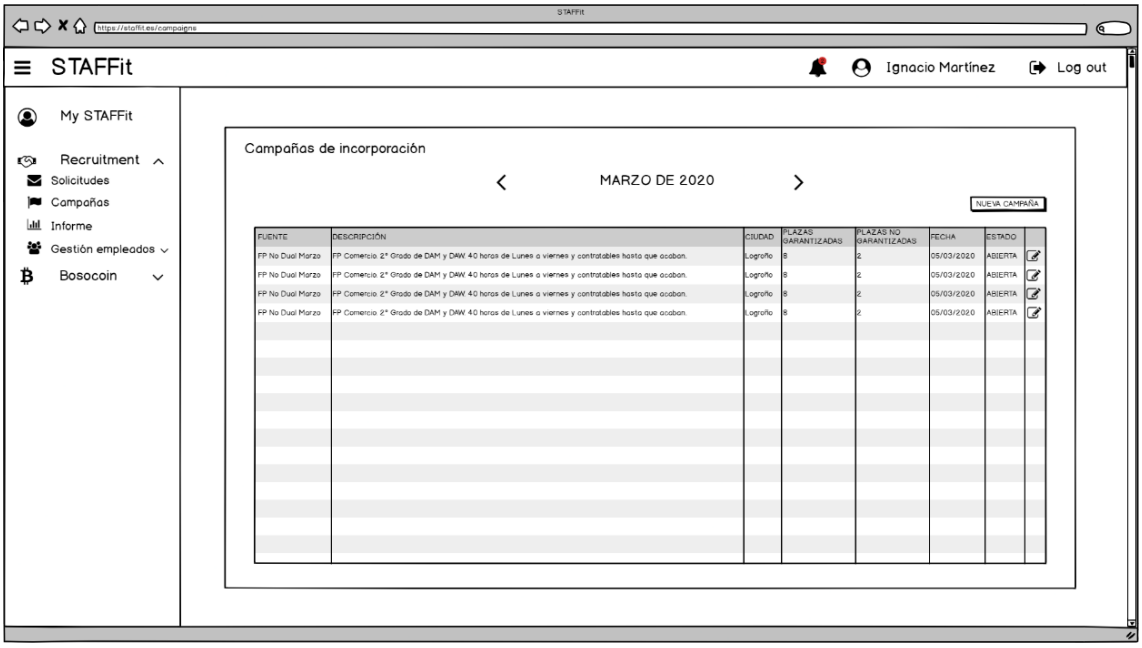


*Figura 5.4.2. Prototipo pantalla de campañas de incorporación*

El apartado *Informe* solo podrá ser accedido por los administradores. En este apartado se mostrará un informe realizado por otros compañeros de la empresa con la herramienta PowerBI. Este informe contará con una serie de estadísticas y gráficas relativas a las solicitudes y campañas de incorporación creadas dentro de STAFFit para poder hacer un estudio de su progreso.

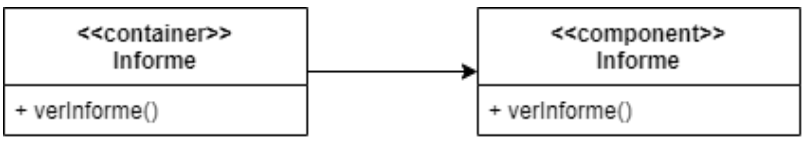


*Figura 5.5.1. Estructura lógica pantalla de informe*

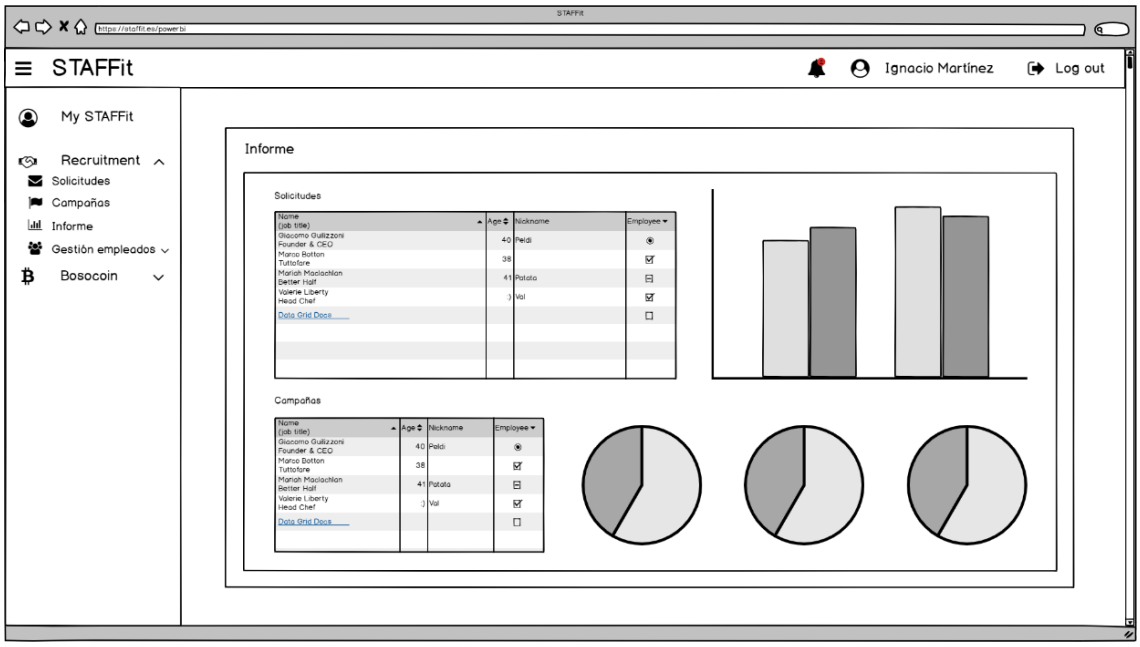


*Figura 5.5.2. Prototipo pantalla de informe*

El último apartado de este módulo corresponde a la gestión de los empleados, e igual que los anteriores solo es accesible por los administradores. Contará con tres subapartados: *Alta, Baja* y *Listado*.

En el subapartado *Alta* se podrá acceder a un formulario por pasos en el que el administrador podrá dar de alta a un nuevo empleado.

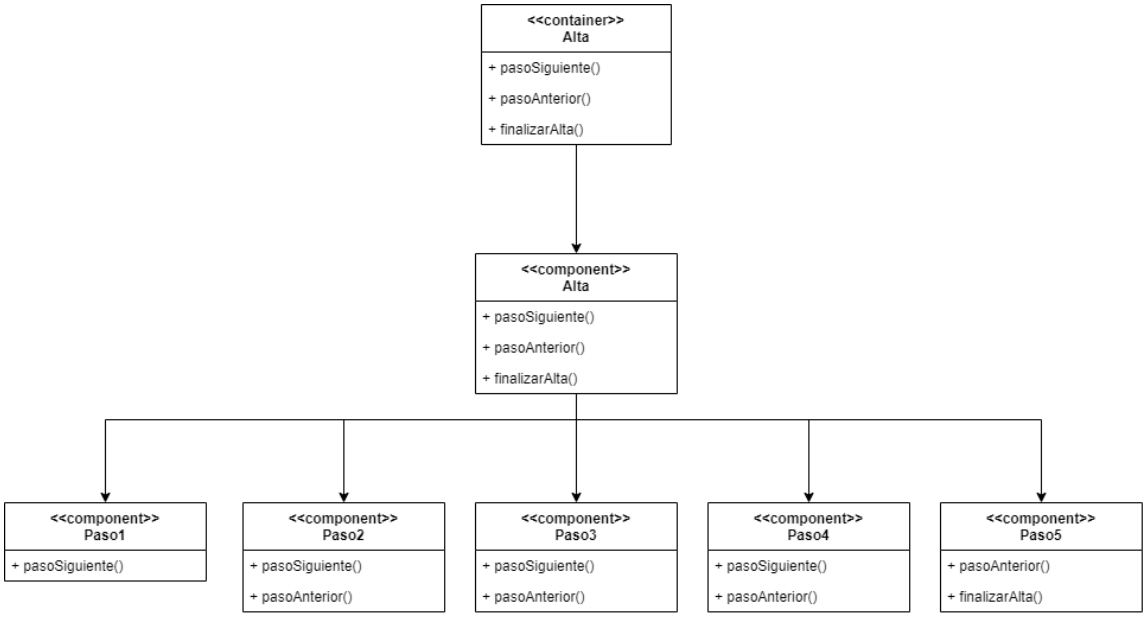


*Figura 5.6.1. Estructura lógica pantalla de alta de empleado*

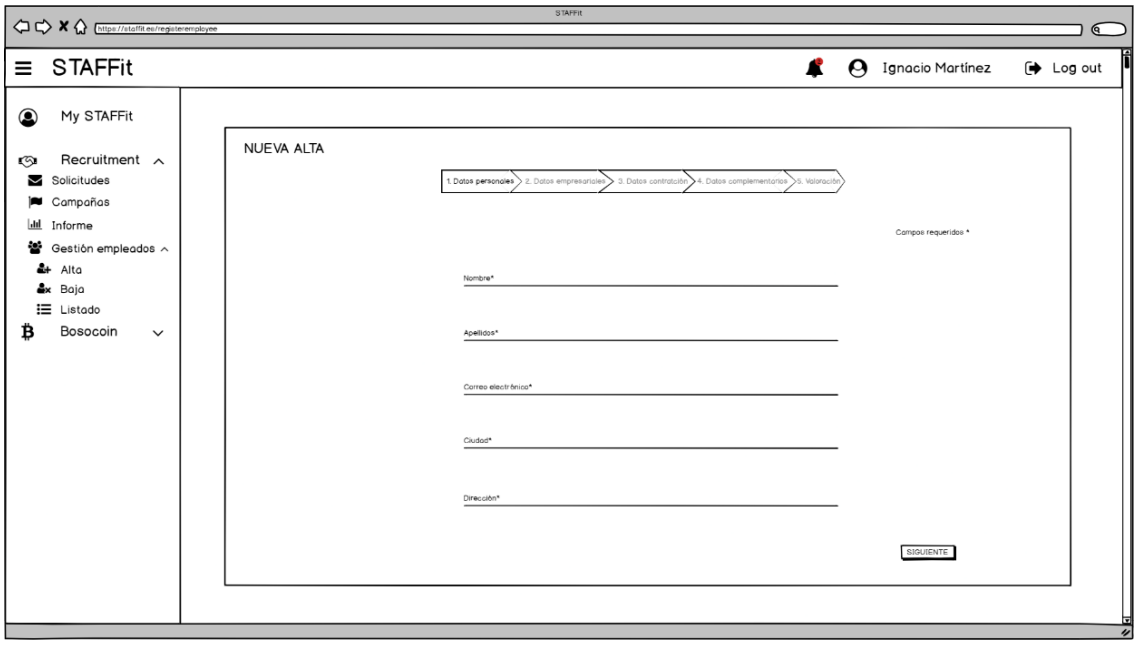


*Figura 5.6.2. Prototipo pantalla de alta de empleado*

En el subapartado *Baja* se mostrará un listado con todos los empleados de la empresa y un buscador para filtrar los resultados de la tabla por nombre. Cada fila de la tabla dispondrá de una casilla para seleccionar al empleado que se desea dar de baja. Una vez se haya seleccionado al empleado se podrá seguir con el proceso de baja de empleado con el enlace disponible en la parte inferior.

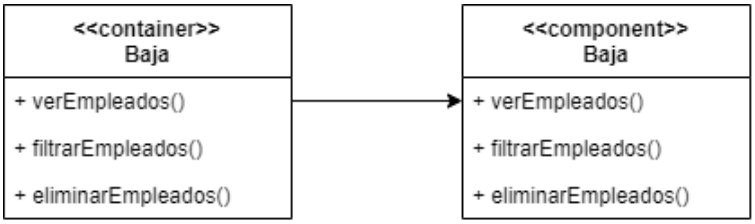


*Figura 5.7.1. Prototipo pantalla de baja de empleado*

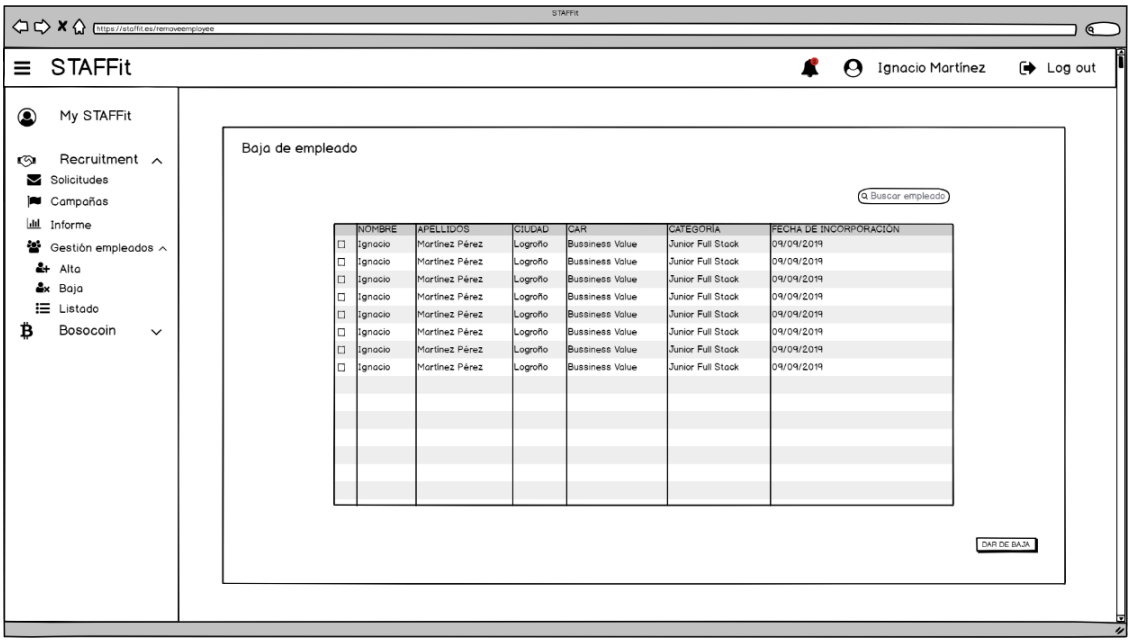


*Figura 5.7.2. Prototipo pantalla de baja de empleado*

El subapartado *Listado* será muy similar al anterior, ya que también mostrará una tabla con todos los empleados de la empresa y dispondrá de un buscador por nombre para filtrar los resultados de la tabla. Cada fila de la tabla contará con un enlace que permitirá acceder a un formulario para editar la información del empleado seleccionado.


*Figura 5.8.2. Estructura lógica pantalla de listado de empleados*

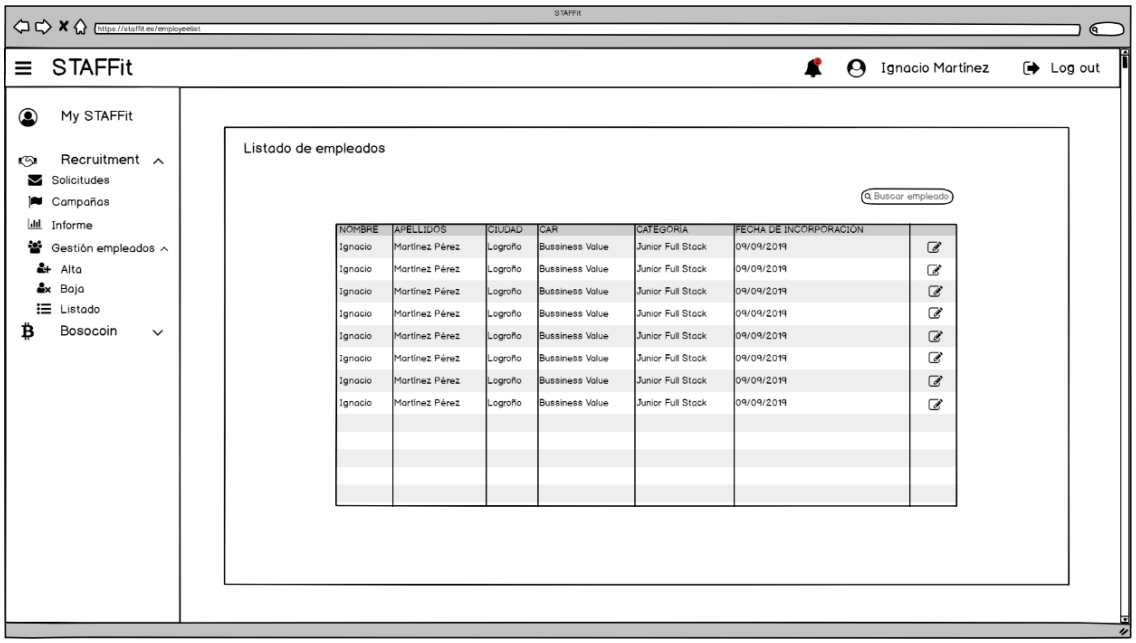


*Figura 5.8.1. Prototipo pantalla de listado de empleados*

## Módulo criptomonedas

El módulo de criptomonedas tendrá dos apartados: *Administración* y *My Wallet.*

El apartado de administración solo será accesible por los administradores. Contará con tres secciones: *Bosocoins en circulación, Historial de transacciones* y *Cartera empresarial.* En *Bosocoins en circulación* se mostrará una gráfica con la evolución temporal de las criptomonedas totales en circulación, así como el balance actual. En *Historial de transacciones* se podrán visualizar todas las transacciones de criptomonedas realizadas por los empleados, pudiéndose filtrar por el nombre del empleado. Cada transacción dispondrá de un enlace que te redirigirá a la herramienta *Etherscan*[2] para ver información más detallada de la transacción. En *Cartera empresarial* se mostrará una gráfica con la evolución temporal del balance de la cartera de criptomonedas de la empresa, así como el balance actual. Dispondrá de un generador de criptomonedas, que permitirá crear nuevas criptomonedas que se enviarán directamente a la cartera empresarial. También se podrán repartir las criptomonedas de la cartera empresarial entre los empleados desde una tabla en la que se mostrarán todos los empleados que disponen de una cartera y su balance actual. Cada fila de la tabla contará con un enlace que permitirá acceder a un sencillo formulario en el que el administrador podrá enviar las criptomonedas que quiera al empleado seleccionado.

[2] *Etherscan* es una herramienta web que te permite explorar la blockchain de Ethereum y ver información detallada de las transacciones y otras actividades llevadas a cabo en Ethereum.

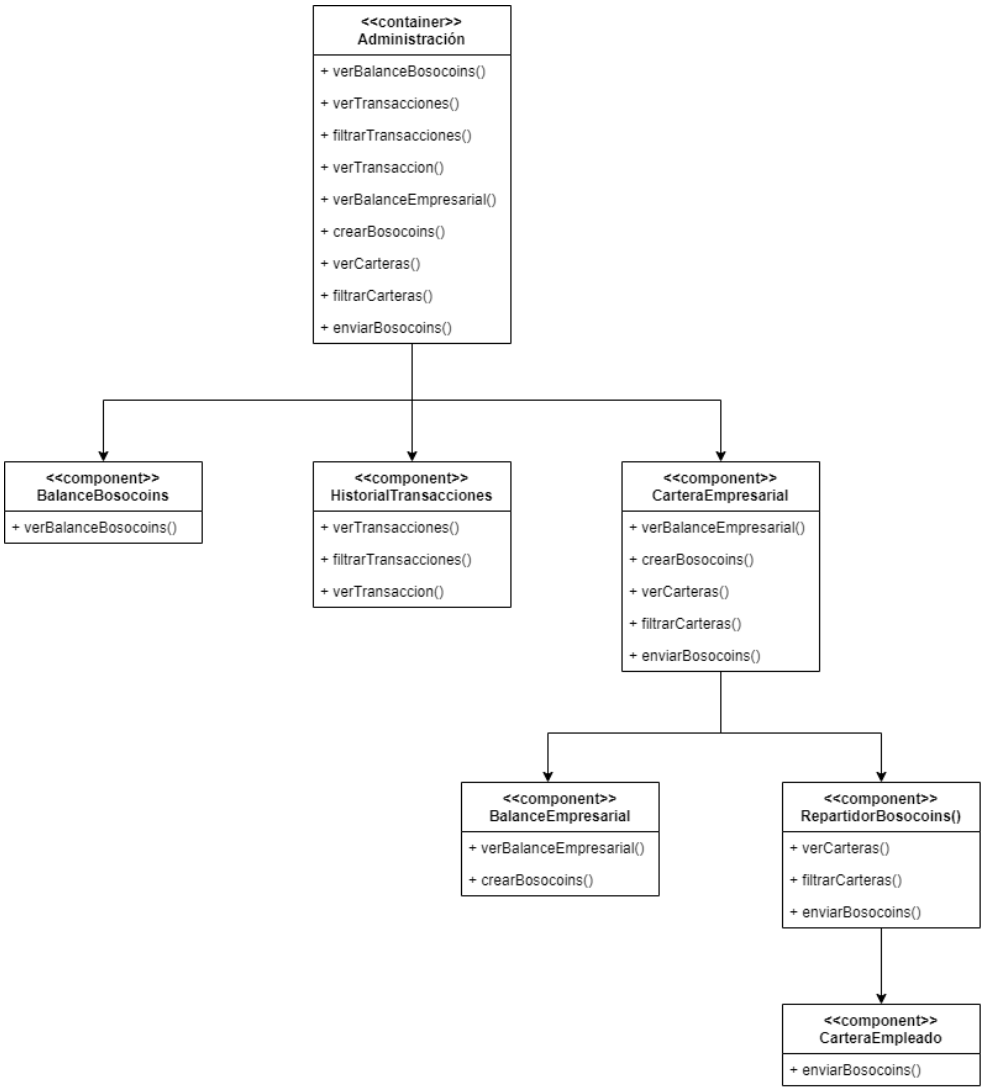


*Figura 5.9.1. Estructura lógica pantalla de administración de la criptomoneda*


*Figura 5.9.2. Prototipo pantalla de administración de la criptomoneda*

El apartado *My Wallet* permitirá acceder a toda la información disponible de la cartera personal del empleado y le permitirá realizar diferentes operaciones con sus criptomonedas. Contará con tres secciones: *My Wallet, Historial de transacciones* y *Transacciones.* En *My Wallet* el empleado podrá ver el estado actual de su cartera, así como la evolución de su balance en el tiempo. En *Historial de transacciones* el empleado podrá ver un listado de sus transacciones, pudiendo filtrar entre las transacciones enviadas y recibidas. También contará con un buscador para filtrar las transacciones por emisor o receptor de la transacción. Al igual que en el apartado anterior, cada transacción del historial contará con un enlace a *Etherscan* en la que el empleado podrá ver información más detallada de la transacción seleccionada. Por último, en *Transacciones* el empleado podrá realizar operaciones con sus criptomonedas. Dispondrá de una tabla de empleados en la que podrá buscar un empleado y enviarle criptomonedas, y de otra tabla con los productos y servicios que ofrece la empresa para canjear tus criptomonedas.

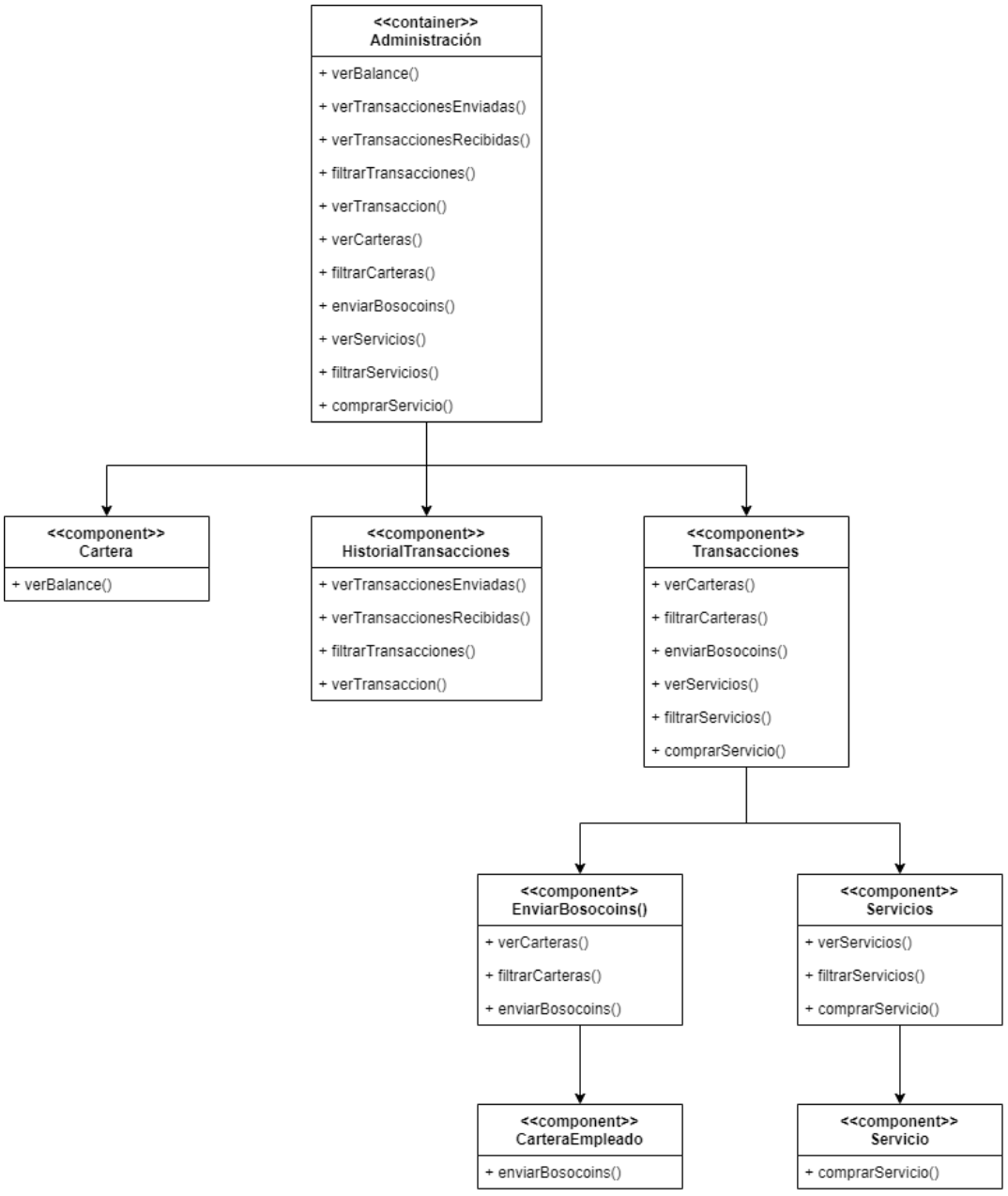


*Figura 5.10.1. Estructura lógica pantalla de cartera de criptomonedas*

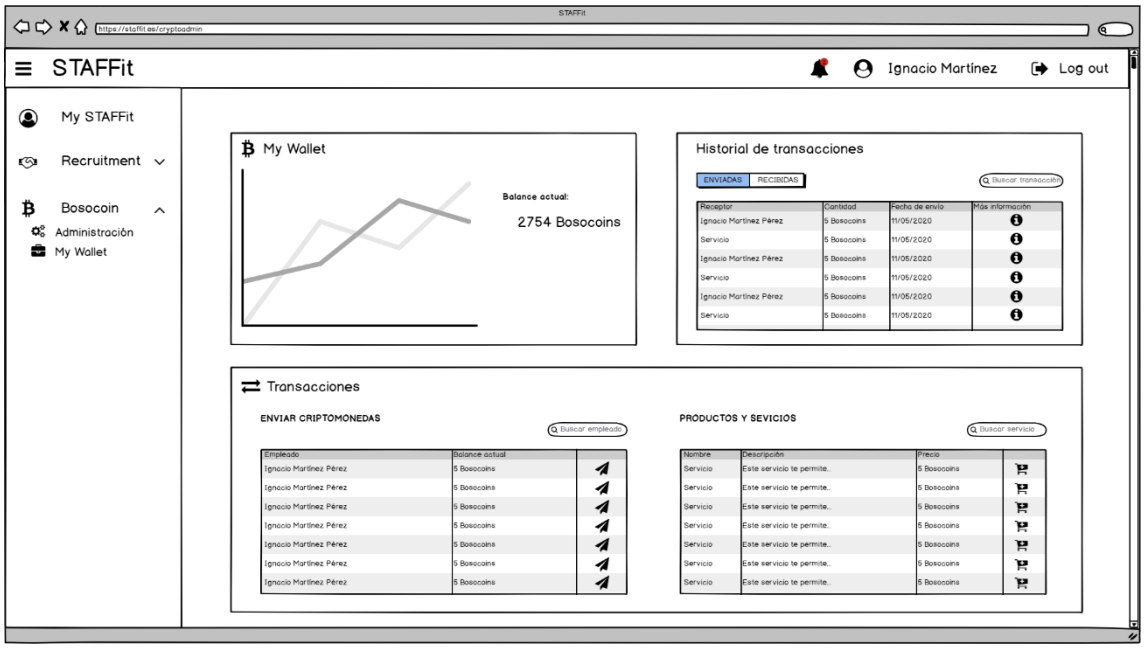


*Figura 5.10.2. Prototipo pantalla de cartera de criptomonedas*

# 5. IMPLEMENTACIÓN

En esta sección se explicará el proceso que se ha seguido para implementar cada una de las partes de nuestra aplicación. Las tecnologías que se nombrarán en adelante se han explicado más detalladamente en la sección *2.2. Tecnologías.*

## 5.1. Construcción de la base de datos y de la API GraphQL

La base de datos de nuestra aplicación se ha creado en PostgreSQL, y la API GraphQL para conectarnos a la misma se ha construido con Hasura GraphQL Engine.

Hasura nos permite crear tanto la base de datos como la API fácilmente y desplegarlos en el servicio en la nube Heroku utilizando su plan gratuito. Una vez que hemos desplegado nuestra aplicación, dispondremos de un enlace con el que podemos acceder a una interfaz gráfica ofrecida por Hasura. Desde esta interfaz podemos construir las tablas de nuestra base de datos directamente, y Hasura se encargará de crear las operaciones GraphQL para las mismas automáticamente. Esto facilita enormemente la creación del back-end de nuestra aplicación, ya que construir tablas o modificarlas es muy intuitivo y la creación de la API GraphQL es automática.

Desde esta interfaz también podemos restringir el acceso a las tablas de nuestra base de datos distinguiendo por roles de usuario. Esto es especialmente relevante, ya que podremos controlar el acceso a nivel de dato de los tres roles de usuario que existen en nuestra aplicación: administrador, mánager y empleado.

Con Hasura podemos configurar eventos que se ejecutarán cuando ocurra un evento determinado en nuestra base de datos mediante la llamada a funciones serverless. La finalidad de esta funcionalidad es poder crear una lógica de negocio más compleja a la que nos ofrece la API GraphQL construida por Hasura. Las funciones serverless que utiliza nuestra aplicación son pequeñas aplicaciones Node.js creadas en el servicio en la nueve Glitch.

A continuación, se muestra un ejemplo de una función serverless. Esta función se ejecutará cuando un administrador haya dado de alta a un empleado desde nuestra aplicación, enviando un correo electrónico con la información de dicho empleado al departamento de recursos humanos.

```
const app = express();

let transporter = nodemailer.createTransport({
  service: "gmail",
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASS
  }
});

app.use(bodyParser.json());

app.post("/", (req, res) => {
  cors(req, res, () => {
    echo(req.body.event).then(incorporation => {
      console.log({ incorporation });

      const dest = process.env.EMAIL_DESTINY;
      const mailOptions = {
        from: `STAFFit <${process.env.EMAIL_USER}>`,
        to: dest,
        subject: "Nueva incorporación",
        html: `<p>NUEVA INCORPORACIÓN:</p>
          <p>Compañeros, os paso la nueva incorporación. Comprobad si los datos son correctos.</p>
          <ul>
            <li>Nombre: ${incorporation.name_employee}</li>
            <li>Apellidos: ${incorporation.surname_employee}</li>
            <li>Correo: ${incorporation.email}</li>
            <li>Teléfono: ${incorporation.tlfn}</li>
            <li>Fecha de incorporación: ${incorporation.incorporation_date}</li>
            <li>Empresa: ${incorporation.company.Empresa}</li>
            <li>CAR: ${incorporation.cars.Descripcion}</li>
            <li>Categoría: ${incorporation.Categoria.DescriCategoria}</li>
            <li>Ciudad: ${incorporation.city.name}</li>
            <li>Jornada: ${incorporation.journey}</li>
            <li>Remuneración fija: ${incorporation.fixed_rate}</li>
            <li>Remuneración variable: ${incorporation.variable_rate || ""}</li>
            <li>Estado: ${incorporation.incorporation_status.name}</li>
          </ul>
          `
      };

      return transporter.sendMail(mailOptions, (erro, info) => {
        if (erro) {
          return res.send(erro.toString());
        }
        return res.send(`Sended`);
      });
    });
  });
});
```

*Figura 6. Función serverless desarrollada en Node.js*

## 5.2. Implementación de la gestión de autenticación

Nuestra aplicación utiliza Auth0 para gestionar la autenticación del usuario y el rol que va a tener dentro de la misma. Auth0 nos permite utilizar los sistemas de autenticación de diferentes proveedores para que los usuarios puedan acceder a nuestra aplicación sin necesidad de registrarse. En nuestro caso, solo hemos activado el sistema de autenticación de Google, ya que los correos corporativos de los empleados utilizan este sistema.

Ya hemos comentado que nuestra aplicación va a tener tres roles diferentes que se van a utilizar tanto en el front-end para restringir el acceso de algunas secciones como en el back-end para

controlar el nivel de acceso a las tablas de nuestra base de datos. Estos roles son asignados al usuario cuando se autentica.

Para implementar esta funcionalidad se ha creado una regla personalizada en Auth0. Una regla de Auth0 es una función escrita en JavaScript que se ejecuta cuando el usuario se autentica y que permite extender las capacidades de Auth0. Los datos de sesión y el rol que se ha asignado al usuario son guardados dentro de un token JWT que se envía al front-end. Un token JWT es un objeto que guarda datos encriptados y utiliza una firma para que no puedan ser modificados.

A continuación, se muestra el código de la regla Auth0 que hemos utilizado en nuestra aplicación. Lo que hace esta regla es comprobar que el correo con el que está intentando acceder el usuario usa el dominio de la empresa (@nter.es o @bosonit.com) y después le asigna el rol que le corresponde.

```
function (user, context, callback) {
  if (!user.email) {
    return callback(new UnauthorizedError('Access denied.'));
  }

  const whitelist = ['nter.es', 'bosonit.com']; //authorized domains

  const userHasAccess = whitelist.some(
      function (domain) {
        const emailSplit = user.email.split('@');
        return emailSplit[emailSplit.length - 1].toLowerCase() === domain;
      }
  );

   if (!userHasAccess) {
    return callback(new UnauthorizedError('Access denied.'));
  }

  let role = 'employee';

  const managers = ['correos de usuarios con rol mánager'];

  if (managers.includes(user.email)) {
    role = 'user';
  }

  const admins = ['correos de usuarios con rol administrador'];

  if (admins.includes(user.email)) {
    role = 'admin';
  }

  const namespace = "https://hasura.io/jwt/claims";

  context.idToken[namespace] = {
    "x-hasura-default-role": role,
    "x-hasura-allowed-roles": ['user', 'admin', 'employee'],
    "x-hasura-user-id": user.email
  };

  return callback(null, user, context);
}
```

*Figura 8. Regla para la autenticación del usuario con Auth0*

## 5.3. Implementación del front-end

### Construcción de la interfaz de usuario

Para la construcción de la interfaz de usuario de nuestra aplicación se ha utilizado la librería de JavaScript React. Crear una aplicación con React es muy sencillo. Necesitamos tener instalado en nuestro equipo tanto Node.js como su gestor de paquetes npm. Una tenemos instalados estos dos elementos, podemos crear y ejecutar nuestra aplicación en React mediante los siguientes comandos:

```
npx create-react-app my-app
cd my-app
npm start
```

Además de React, hemos utilizado más librerías para la construcción de la interfaz de usuario. Una de las principales ha sido Material UI, que nos permite implementar el patrón de diseño Material de Google en nuestras vistas y nos ha facilitado las tareas de maquetación. Para poder utilizar Material UI dentro de nuestra aplicación deberemos instalarlo en nuestro proyecto con herramienta npm. Para ello simplemente tendremos que ejecutar el siguiente comando desde la raíz del directorio de nuestro proyecto:

```
npm install @material-ui/core
```

React está orientado a la construcción de vistas mediante componentes. Un componente es un elemento independiente y reutilizable que nos permite separar nuestra interfaz por piezas. A continuación, se muestra el código de dos de los componentes utilizados en nuestra aplicación.

El primer fragmento de código corresponde al componente que representa la tabla que muestra las campañas de incorporación. Podemos ver que a su vez emplea componentes propios de la librería Material UI (Table, TableBody, TableContainer, etc.), que se utilizan para construir nuestra tabla de datos con el patrón de diseño Material. También podemos ver que emplea el componente CampaignItem, el cual corresponde al segundo fragmento de código y representa cada una de las filas de la tabla de campañas.

```jsx
import React from 'react';
import { CampaignItem } from './CampaignItem';
import './CampaignList.css';
import {
  Table,
  TableBody,
  TableContainer,
  TableHead,
  TableRow,
  Paper,
} from "@material-ui/core";
import { StyledTableCell } from "../components/StyledTableCell";
import { withStyles, makeStyles } from "@material-ui/core/styles";

const StyledTableRow = withStyles(theme => ({
  root: {
    "&:nth-of-type(odd)": {
      backgroundColor: theme.palette.background.default
    }
  }
}))(TableRow);

const useStyles = makeStyles({
  table: {
    minWidth: 700
  }
});

export function CampaignList({ data, refetch, edit }) {
  const trs = data.campaign.map(item => (
    <CampaignItem
      item={item}
      key={item.id}
      refetch={() => refetch()}
      edit={() => edit(item)}
    />
  ));

  const classes = useStyles();

  return (
    <TableContainer style={{ width: "100%" }} component={Paper} >
      <Table aria-label="customized table"  className={classes.table} >
        <TableHead>
          <StyledTableRow>
            <StyledTableCell align="center">ID</StyledTableCell>
            <StyledTableCell align="center">FUENTE</StyledTableCell>
            <StyledTableCell align="center">DESCRIPCIÓN</StyledTableCell>
            <StyledTableCell align="center">CIUDAD</StyledTableCell>
            <StyledTableCell align="center">PLAZAS ASEGURADAS</StyledTableCell>
            <StyledTableCell align="center">PLAZAS NO ASEGURADAS</StyledTableCell>
            <StyledTableCell align="center">INCORPORACIÓN</StyledTableCell>
            <StyledTableCell align="center">FINALIZACIÓN</StyledTableCell>
            <StyledTableCell align="center">ESTADO</StyledTableCell>
            <StyledTableCell align="center" style={{whiteSpace:'nowrap'}} >ASIGNADO A</StyledTableCell>
            <StyledTableCell></StyledTableCell>
          </StyledTableRow>
        </TableHead>
        <TableBody>{trs}</TableBody>
      </Table>
    </TableContainer>
  );
}
```

*Figura 9.1. Componente React para la tabla de campañas de incorporación*

```jsx
import React from "react";
import { isoDateddMMyyyy } from "../utils/date";
import "react-toastify/dist/ReactToastify.css";
import IconButton from "@material-ui/core/Button";
import { TableRow } from "@material-ui/core";
import { StyledTableCell } from "../components/StyledTableCell";
import EditIcon from "@material-ui/icons/Edit";
import { makeStyles } from "@material-ui/core/styles";

const useStyles = makeStyles(theme => ({
  root: {
    display: "flex",
    "& > *": {
      margin: theme.spacing(1)
    },

    [theme.breakpoints.up(1300)]: {
      fontSize: "1rem"
    },
    [theme.breakpoints.down(1300)]: {
      fontSize: "0.85rem"
    }
  }
}));

export function CampaignItem({ item, refetch, edit }) {
  const classes = useStyles();

  return (
    <TableRow>
      <StyledTableCell align="center">{item.id}</StyledTableCell>
      <StyledTableCell align="center">{item.source}</StyledTableCell>
      <StyledTableCell align="left">{item.description}</StyledTableCell>
      <StyledTableCell align="center">{item.city.name}</StyledTableCell>
      <StyledTableCell align="center">{item.guaranteed_places}</StyledTableCell>
      <StyledTableCell align="center">
        {item.unguaranteed_places}
      </StyledTableCell>
      <StyledTableCell align="center">
        {isoDateddMMyyyy(item.incorporation_date)}
      </StyledTableCell>
      <StyledTableCell align="center">
        {isoDateddMMyyyy(item.finish_date)}
      </StyledTableCell>
      <StyledTableCell align="center">{item.status.name}</StyledTableCell>
      <StyledTableCell align="center">{item.assigned_to}</StyledTableCell>
      <StyledTableCell align="center">
        <IconButton color="default" variant="text" onClick={edit}>
          <EditIcon className={classes.root} />
        </IconButton>
      </StyledTableCell>
    </TableRow>
  );
}
```

*Figura 9.2. Componente React para las filas de la tabla de campañas de incorporación*

Como resultado de los dos componentes anteriores obtenemos la tabla que se muestra en la siguiente imagen.

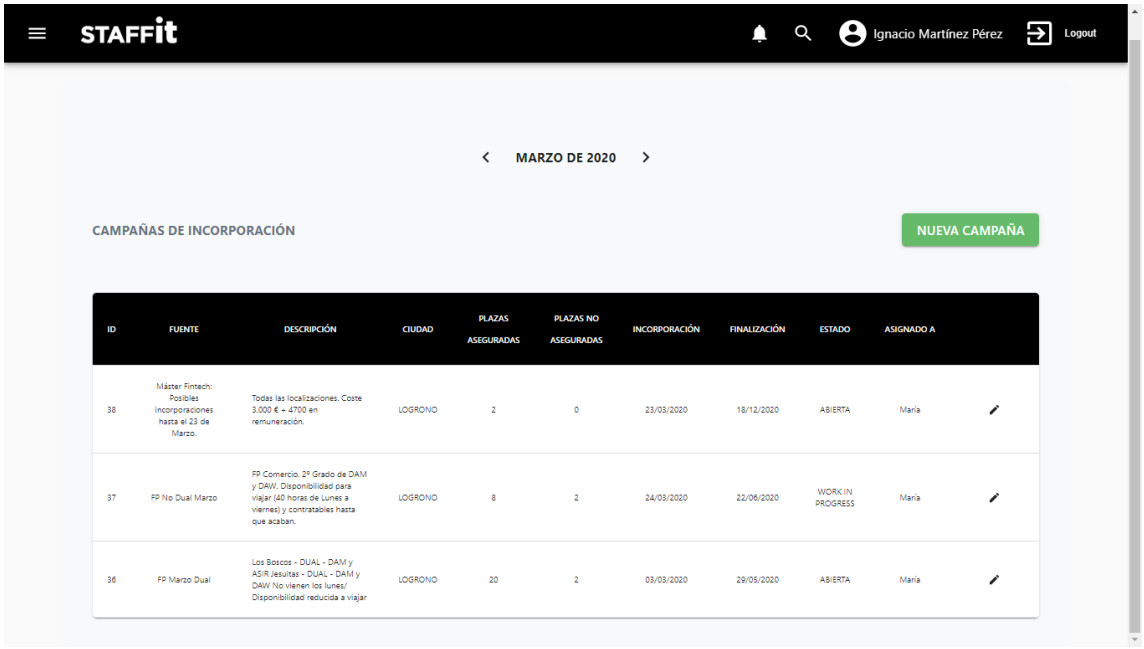


*Figura 9.3. Pantalla de campañas de incorporación*

## Conexión con Auth0

Integrar Auth0 dentro de nuestra aplicación es muy sencillo. Simplemente tenemos que implementar en nuestra aplicación un componente que Auth0 nos facilita en su página web y configurarlo con el dominio e identificador de cliente que Auth0 nos proporciona desde su panel de control.

Una vez que lo hemos integrado, podemos utilizar el token JWT que Auth0 crea cuando un usuario se autentica para comprobar el rol que este tiene y controlar así las pantallas a las que tiene acceso.

En el siguiente fragmento de código se muestra el componente que controla las rutas de nuestra aplicación. Primero se obtiene el token JWT del usuario con la función *useAuth0()* que nos proporciona el componente que hemos implementado anteriormente. Después se extrae el rol del token con el que controlamos a que rutas tiene acceso el usuario. Si el usuario intenta acceder a una ruta a la que tiene el acceso restringido, se le muestra una pantalla de error que hemos definido en el componente *ErrorNoAdmin* y del que podemos ver un ejemplo en la imagen.

```jsx
export default function AppRouter() {
  const { user } = useAuth0();
  const role = user["https://hasura.io/jwt/claims"]["x-hasura-default-role"];

  return (
    <Router>
      <Switch>
        <Route path="/" exact>
          <Login />
        </Route>
        <PrivateRoute path="/administrador">
          {role === "admin" ? <Administrador /> : <ErrorNoAdmin />}
        </PrivateRoute>
        <PrivateRoute path="/demands">
          {role !== "employee" ? <CARView /> : <ErrorNoAdmin />}
        </PrivateRoute>
        <PrivateRoute path="/campaign">
          {role === "admin" ? <Campaign /> : <ErrorNoAdmin />}
        </PrivateRoute>
        <PrivateRoute path="/powerbi">
          {role === "admin" ? <Report /> : <ErrorNoAdmin />}
        </PrivateRoute>
        <PrivateRoute path="/recruitment">
          {role === "admin" ? <Recruitment /> : <ErrorNoAdmin />}
        </PrivateRoute>
        <PrivateRoute path="/recruitmentlist">
          {role === "admin" ? <RecruitmentListPage /> : <ErrorNoAdmin />}
        </PrivateRoute>
        <PrivateRoute path="/unsubscribed">
          {role === "admin" ? <Unsubscribed /> : <ErrorNoAdmin />}
        </PrivateRoute>
        <PrivateRoute path="/mystaffit">
          <MySTAFFit />
        </PrivateRoute>
        <PrivateRoute path="/mymentors">
          <MyMentors />
        </PrivateRoute>
        <PrivateRoute path="/flats">
          {role === "admin" ? <FlatListPage /> : <ErrorNoAdmin />}
        </PrivateRoute>
      </Switch>
    </Router>
  );
}
```

*Figura 10.1. Componente React para el control del acceso por roles*

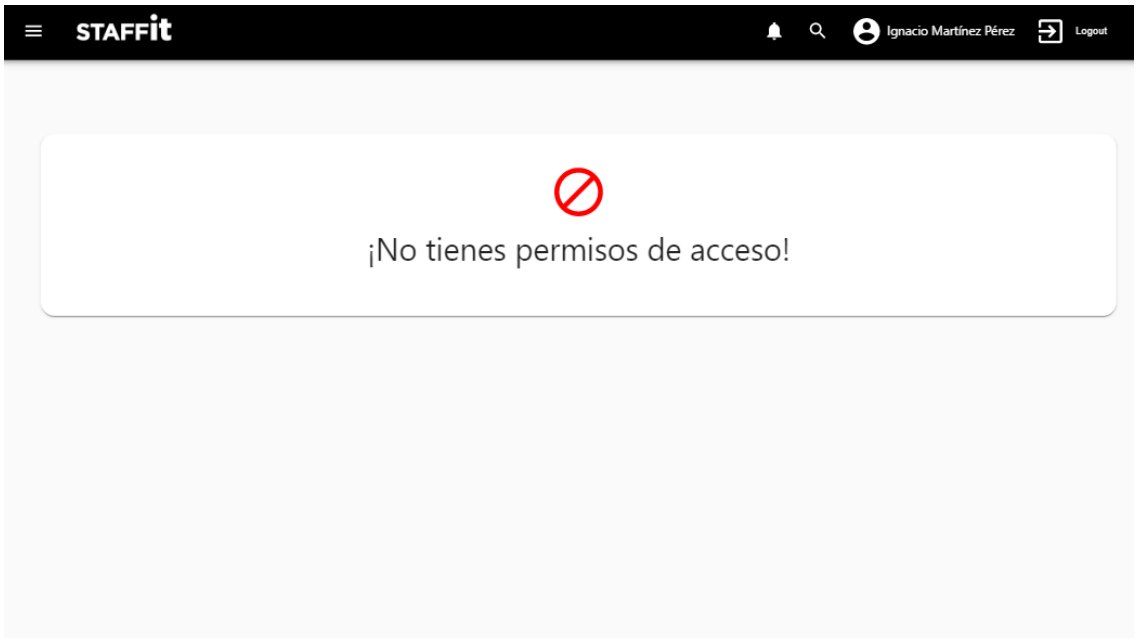


*Figura 10.2. Pantalla de restricción de acceso*

## Conexión con la API GraphQL

Para conectarnos con nuestra API desde React se ha utilizado la librería Apollo. La implementación de Apollo es muy sencilla. Simplemente tenemos que crear un cliente de Apollo al que le indicamos el endpoint de nuestra API y el token JWT del usuario.

```
import ApolloClient from "apollo-client";
import { InMemoryCache } from "apollo-cache-inmemory";
import { HttpLink } from "apollo-link-http";

export const createApolloClient = (authToken) => {
  return new ApolloClient({
    link: new HttpLink({
      uri: process.env.GRAPHQL_ENDPOINT,
      headers: {
        Authorization: `Bearer ${authToken}`,
      },
    }),
    cache: new InMemoryCache(),
  });
};
```

*Figura 11.1. Conexión con API GraphQL mediante Apollo*

Con el token JWT que se envía a la API podemos controlar en el back-end el acceso a nivel de dato que tiene el usuario dependiendo de su rol. Los permisos por rol de usuario para cada tabla de nuestra base de datos se pueden configurar fácilmente desde la interfaz gráfica de Hasura tal cual se ha visto anteriormente en la sección *5.1. Construcción de la base de datos y de la API GraphQL*.

Las llamadas a una API hecha en GraphQL son diferentes a las de una API REST convencional. Tienen tres parámetros principales: tipo de llamada, cuerpo de la llamada y variables de la llamada. Hay dos tipos de llamadas, las consultas y las mutaciones. Las consultas nos permiten traer datos del servidor, mientras que las mutaciones nos permiten insertar, modificar o eliminar datos del servidor.

A continuación, se muestra un ejemplo de una consulta que nos permite obtener las campañas de incorporación cuya fecha de incorporación esta entre dos fechas determinadas. Con *query* indicamos que nuestra llamada es una consulta. Después indicamos el nombre de la operación de la API GraphQL que queremos ejecutar, en este caso *queryCampaign* junto con las variables de entrada que tiene la consulta. Por último, se indica la tabla de la que queremos obtener los datos (*campaign)*, la condición que han de cumplir los datos (*where)* y que campos de la tabla queremos (id, source, city_id, etc.).

```
export const QUERY_CAMPAIGN = gql`
  query queryCampaign($begin: date, $end: date) {
    campaign(where: { incorporation_date: { _gte: $begin, _lte: $end } }) {
      id
      source
      city_id
      guaranteed_places
      unguaranteed_places
      incorporation_date
      finish_date
      description
      status_id
    }
  }
`;
```

*Figura 11.2. Consulta a la tabla de campañas de incorporación con GraphQL*

Para ejecutar la consulta anterior hacemos uso del cliente Apollo que hemos creado anteriormente. Apollo nos proporciona la función *useQuery* para ejecutar consultas, a la que simplemente tenemos que introducirle la consulta anterior y sus variables de entrada.

```
const { loading, error, data } = useQuery(
  QUERY_CAMPAIGN,
  {
    variables: {
      begin: begin.toISOString().substr(0, 10),
      end: end.toISOString().substr(0, 10)
    }
  }
);
```

*Figura 11.3. Llamada a la API con Apollo utilizando una consulta GraphQL*

### Conexión con la criptomoneda

Podremos conectarnos a los smart contracts de nuestra criptomoneda utilizando la dirección que obtendremos cuando los despleguemos en la blockchain y su código compilado. Para ello emplearemos la librería de JavaScript Web3. Lo primero que tenemos que hacer es conectarnos a un proveedor blockchain que nos enlace con la red que deseamos. En nuestro caso vamos a

utilizar el proveedor Infura para conectarnos a la red Ropsten, donde tenemos desplegada la criptomoneda.

Una vez conectados a la red, podremos conectarnos a la criptomoneda utilizando el archivo JSON que obtendremos del smart contract cuando lo compilemos y la dirección en la que ha sido desplegado.

```
const Web3 = require("web3");
const provider = new Web3.providers.HttpProvider(
  "https://ropsten.infura.io/v3/${process.env.INFURA_KEY}"
);
const web3 = new Web3(provider);
const fs = require("fs");

web3.eth.net
  .isListening()
  .then(() => {
    console.log("web3 is connected");
    main();
  })
  .catch((e) => console.log("Something went wrong"));

async function main() {
  const contract = JSON.parse(fs.readFileSync("./Token.json", "utf8"));
  const abi = contract.abi;
  const token_Address = process.env.TOKEN_ADDRESS;
  const token = new web3.eth.Contract(abi, token_Address);
}
```

*Figura 11.4. Conexión con la criptomoneda mediante Web3*

## 5.4. Implementación de la criptomoneda

### Construcción de los smart contracts

Para el desarrollo de los smart contracts de nuestra criptomoneda se ha utilizado el marco de desarrollo Truffle. Para poder trabajar con Truffle en nuestro equipo tenemos que instalarlo con la herramienta npm de manera similar a como hemos instalado las librerías JavaScript para nuestra aplicación React. El comando de instalación es el siguiente:

```
npm install -g truffle
```

Una vez que hemos instalado Truffle en nuestro equipo, podemos inicializar un nuevo proyecto Ethereum con el siguiente comando.

```
truffle init bosocoin
```

La estructura del proyecto que hemos inicializado será la siguiente:

- contracts/: Directorio en el que se almacenan nuestros smart contracts.

- build/: Directorio en el que se almacena el código compilado de nuestros smart contracts.
- migrations/: Directorio en el que se almacenan los scripts para el despliegue de lo smart contracts.
- test/: Directorio en el que se almacenan los test unitarios para nuestros smart contracts.


- truffle-config.js: Archivo donde se guarda los datos de configuración de nuestro proyecto (versión del compilador, red en la que se desplegarán los smart contracts, etc.).

El lenguaje que se ha utilizado para el desarrollo de los smart contracts es Solidity. Solidity es un lenguaje orientado a contratos. Un contrato es bastante similar a una clase en un lenguaje orientado a objetos, ya que podemos crear instancias de un contrato al igual que creamos instancias de una clase.

A continuación, se muestra el código de uno de los smart contracts de nuestra criptomoneda. Tenemos un contrato *TokenFactory* con una única función *createToken*, que se encarga de crear una nueva criptomoneda creando una nueva instancia de otro de nuestros contratos llamado *Token* con los parámetros de entrada que le hemos indicado. Una vez que se ha creado la nueva criptomoneda, se emite un evento *TokenCreated* que podemos consultar externamente para comprobar que nuestra criptomoneda se ha creado correctamente.

```
pragma solidity ^0.6.0;

import "./Token.sol";
import "./libraries/SafeMath.sol";

contract TokenFactory {
    // Using SafeMath library to prevent overflows and underflows
    using SafeMath for uint256;

    mapping(string => address) public tokens;
    uint256 public numTokens = 0;

    event TokenCreated(
        address indexed addr,
        string name,
        string symbol,
        uint8 decimals,
        uint256 initialSupply,
        address indexed owner
    );

    /// @dev Deploy an ERC20 token contract, register it with TokenRegistry,
    ///      and returns the new token's address.
    /// @param _name The name of the token
    /// @param _symbol The symbol of the token.
    /// @param _decimals The decimals of the token.
    /// @param _initialSupply The initial supply of the token.
    function createToken(
        uint256 _initialSupply,
        uint8 _decimals,
        string calldata _name,
        string calldata _symbol
    ) external returns (address addr) {
        require(tokens[_symbol] == address(0), "A token with the same symbol already exists!");

        Token token = new Token(_initialSupply, _decimals, _name, _symbol, msg.sender);

        addr = address(token);
        tokens[_symbol] = addr;
        numTokens = numTokens.add(1);

        emit TokenCreated(addr, _name, _symbol, _decimals, _initialSupply, msg.sender);
    }
}
```

*Figura 12.1. Smart contract para la creación de criptomonedas*

## Despliegue de los smart contracts

Al igual que hemos utilizado Truffle para la construcción de los smart contracts, también se puede emplear para desplegarlos en la red blockchain que deseemos. Como ya se ha explicado en la sección *4.1. Arquitectura*, los smart contracts de nuestra criptomoneda están desplegados en la red de pruebas Ropsten.

Siguiendo con el ejemplo del contrato mostrado anteriormente, vamos a explicar el procedimiento que se ha seguido para desplegar dicho smart contract.

Primero debemos configurar Truffle para que despliegue nuestros smart contracts sobre la red Ropsten. Esta configuración se lleva a cabo en el *truffle-config.js* de nuestro proyecto. Después

se crea un script de despliegue del smart contract escrito en JavaScript como se muestra a continuación.

```
const TokenFactory = artifacts.require("TokenFactory");

module.exports = function (deployer) {
  deployer.deploy(TokenFactory);
};
```

*Figura 12.2. Script de despliegue de un smart contract*

Por último, desplegamos nuestro smart contract con el siguiente comando:

```
truffle deploy –network ropsten
```

Cuando desplegamos un smart contract obtenemos como resultado la dirección que éste tiene dentro de la red blockchain. Esta dirección es la que empleamos posteriormente para conectarnos externamente con el smart contract y poder así hacer transacciones con el mismo.

# 6. PRUEBAS

Para comprobar el correcto funcionamiento de nuestra aplicación ha sido necesario realizar las pruebas pertinentes. Todas las pruebas que se han realizado se han llevado a cabo en la fase final de implementación de cada funcionalidad, de acuerdo con la metodología iterativo-incremental que hemos seguido para el desarrollo de nuestro proyecto.

Podemos dividir las pruebas que se han realizado en dos grandes grupos:

1. Test unitarios. Empleados para verificar que no hay errores en los smart contracts de nuestra criptomoneda.
2. Pruebas de integración. Realizadas para comprobar el correcto funcionamiento de la aplicación web.

## 6.1. Test unitarios

Los smart contracts son inmutables una vez han sido desplegados en la red blockchain. Esto permite a los usuarios estar seguros de que las reglas por las cuales sus criptomonedas son operadas no serán cambiadas. Sin embargo, esta característica hace que desarrollar smart contracts sea especialmente delicado. Si desplegamos un smart contract con un error o vulnerabilidad estará hay para siempre, por lo que realizar test unitarios para los smart contracts es especialmente importante.

Al igual que en las fases de desarrollo y despliegue de los smart contracts, también hemos usado Truffle para la fase de testeo. Lo primero que tenemos que hacer para poder testear nuestros smart contracts es crear una blockchain de desarrollo en la que desplegar y testear los smart contracts antes de desplegarlos en una red blockchain de producción. Para ello se ha utilizado Ganache, una herramienta que pertenece a la suite de Truffle y que nos permite crear una blockchain privada de desarrollo en local de manera sencilla.

Ganache nos ofrece una interfaz desde la que podemos crear espacios de trabajo rápidamente y ver los smart contracts que tenemos desplegados en local. Una vez hemos creado nuestro espacio de trabajo, desplegamos los smart contracts en Ganache usando Truffle. El proceso es similar al que hemo explicado anteriormente. Configuramos el archivo *truffle-config.js* con la red privada que hemos creado con Ganache y ejecutamos el siguiente comando.

```
truffle deploy –network development
```

Cuando se hayan desplegado los smart contracts podremos visualizarlos en la interfaz de Ganache.

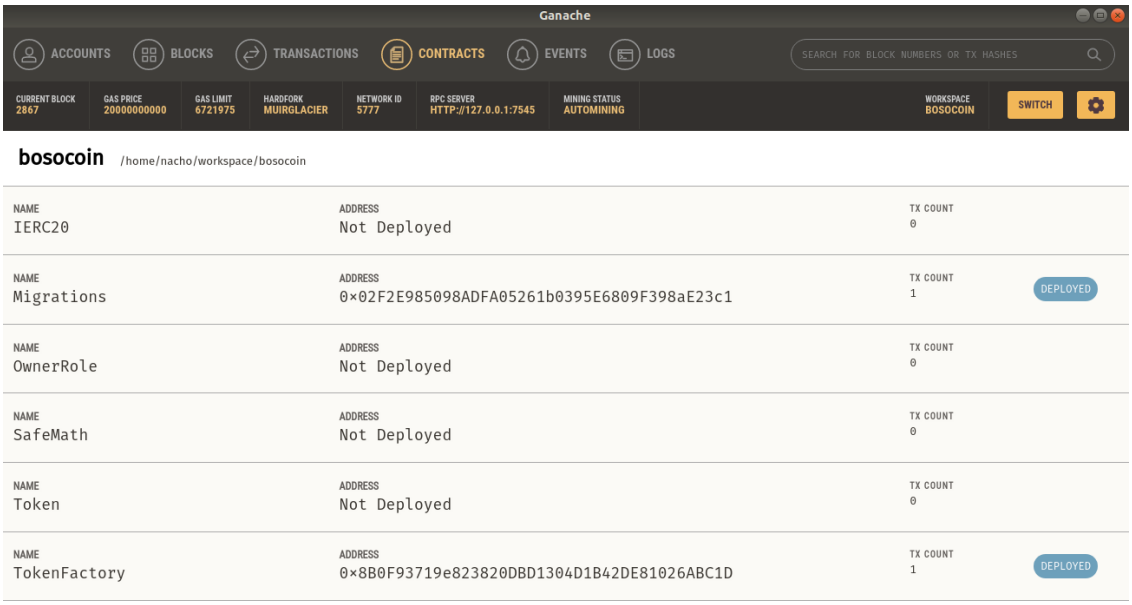


*Figura 13.1. Visualización de los smart contracts en la interfaz de Ganache*

Ahora que ya tenemos un entorno en el que podemos testear los smart contracts, podemos crear los test unitarios. Los test unitarios que hemos utilizado se han desarrollado en JavaScript. A continuación, se muestra un ejemplo de un test unitario que hemos utilizado para comprobar que nuestro contrato *TokenFactory* permite crear una nueva criptomoneda de correctamente con los parámetros que le hemos indicado.

```
const TokenFactory = artifacts.require("TokenFactory");
const Token = artifacts.require("Token");
const truffleAssert = require("truffle-assertions");

contract("TokenFactory", function (accounts) {
  const initialSupply = 10;
  const decimals = 1;
  const name = "Boson";
  const symbol = "BSN";
  const owner = accounts[0];

  beforeEach(async () => {
    factory = await TokenFactory.new();
  });

  it("should be able to create a new token and make the sender the owner of it", async () => {
    const result = await factory.createToken(
      initialSupply,
      decimals,
      name,
      symbol
    );
    const tokenAddress = await factory.tokens(symbol);

    assert.equal(result.receipt.status, true);
    assert.equal(result.logs[0].args.addr, tokenAddress);
    assert.equal(result.logs[0].args.name, name);
    assert.equal(result.logs[0].args.symbol, symbol);
    assert.equal(result.logs[0].args.decimals, decimals);
    assert.equal(result.logs[0].args.initialSupply, initialSupply);
    assert.equal(result.logs[0].args.owner, owner);
    assert.equal(await factory.numTokens(), 1);

    const newToken = await Token.at(tokenAddress);

    assert.equal(await newToken.owners(owner), true);
  });
});
```

*Figura 13.2. Test unitario para la verificación de un smart contract*

Para ejecutar el test anterior con Truffle, se utiliza el siguiente comando, donde *./test/token_factory.js* es la ruta en la que se encuentra el archivo de nuestro test.

```
truffle test ./test/token_factory.js
```

## 6.2. Pruebas de integración

Para comprobar el buen funcionamiento de nuestra aplicación web no se han llevado a cabo test unitarios como con los smart contracts, sino que se han realizado pruebas directamente en local antes del despliegue de cada nueva funcionalidad. Para llevar a cabo estas pruebas era necesario crear un back-end de desarrollo para poder testear las diferentes operaciones sin modificar nuestra base de datos de producción.

Por ello se decidió crear un entorno de desarrollo mediante Docker, una herramienta que te permite desplegar aplicaciones dentro de contenedores que pueden ser ejecutados en cualquier equipo. Este entorno de desarrollo consiste en un contenedor Docker que empaqueta una base

de datos PostgreSQL y una API creado con Hasura similar a las que tenemos en nuestro entorno de producción, y que tenemos desplegado en nuestro equipo para poder hacer las pruebas directamente en local.

Para construir un contenedor Docker tenemos que crear un archivo de configuración llamado *docker-compose.yaml* en el que indicamos los servicios que empaquetará el contenedor y sus propiedades. En nuestro caso, este archivo contiene esta información:

```
version: '3.6'
services:
  postgres:
    image: postgres
    restart: always
    environment:
      POSTGRES_PASSWORD: "admin"
    volumes:
    - db_data:/var/lib/postgresql/data
  graphql-engine:
    image: hasura/graphql-engine:v1.1.0
    ports:
    - "8080:8080"
    depends_on:
    - "postgres"
    restart: always
    environment:
      HASURA_GRAPHQL_DATABASE_URL: postgres://postgres:admin@postgres:5432/postgres
      HASURA_GRAPHQL_ENABLE_CONSOLE: "true"
      HASURA_GRAPHQL_ENABLED_LOG_TYPES: startup, http-log, webhook-log, websocket-log, query-log
      HASURA_GRAPHQL_ADMIN_SECRET: "admin"
      HASURA_GRAPHQL_JWT_SECRET: '{"type": "RS512", "key": "-----BEGIN CERTIFICATE-----\nMI… "}'
volumes:
  db_data:
```

*Figura 14. Archivo de configuración del contenedor Docker para la creación de nuestro entorno de desarrollo*

Una vez que hemos desplegado nuestro contenedor, podremos acceder desde local a una interfaz de Hasura similar a la mostrada en la sección *5.1. Construcción de la base de datos y de la API GraphQL*.

# 7. SEGUIMIENTO Y CONTROL

## 7.1. Riesgos materializados

El mayor cambio que ha sufrido el proyecto ocurrió a finales de abril, cuando la empresa para la que se iba a desarrollar esta aplicación cambio totalmente el alcance del proyecto y su dirección. La empresa quería crear una aplicación con muchas más funcionalidades que las propuestas inicialmente además de utilizar una arquitectura completamente diferente a la que se estaba utilizando hasta el momento. Este gran cambio requería empezar el proyecto desde cero, y era un riesgo que no se había llegado a contemplar en el plan de contingencia.

Ante esta situación se decidió que la mejor alternativa era continuar con el desarrollo del proyecto siguiendo la propuesta inicial de la empresa. Este proyecto no va a ser utilizado por la empresa, sino que es un proyecto totalmente independiente.

Este gran cambio junto con la falta de tiempo han sido los motivos por los cuales se han decidido recortar el alcance del proyecto tal cual habíamos contemplado en nuestro plan de contingencia.

## 7.2. Objetivos alcanzados

Los objetivos principales del proyecto se han cumplido. Se ha conseguido desarrollar una aplicación que unifica la gestión de recursos humanos de una empresa y un portal de empleados con un sistema de incentivos mediante criptomonedas en una plataforma.

La aplicación cuenta con un control por roles de usuario tanto a nivel de interfaz de usuario como a nivel de acceso a la base de datos. Además, se ha logrado implementar los tres módulos propuestos inicialmente: módulo de gestión de recursos humanos, módulo de empleado y módulo de criptomonedas.

Todos los requisitos del módulo de gestión de recursos humanos y del módulo de empleado han sido alcanzados, y la mayoría de los requisitos del módulo de criptomonedas también han sido alcanzados.

## 7.3. Objetivos no alcanzados

Los objetivos no alcanzados han sido causados por el recorte del alcance producido por los riesgos que hemos explicado anteriormente, por el que decidimos no implementar algunos de los requisitos.

Este cambio de alcance ha afectado principalmente al módulo de criptomonedas. Finalmente, nuestra aplicación no permite canjear criptomonedas por servicios o productos. El motivo por el que se decidió no implementar este requisito fue porque en la propuesta inicial de la empresa para este proyecto no se llegaron a definir los productos y servicios que se podrían comprar en la aplicación, y tras decidir continuar con el proyecto de manera independiente ya no era una funcionalidad prioritaria.

También ha afectado al módulo de gestión de recursos humanos, ya que finalmente no se ha implementado la funcionalidad para dar de baja a un empleado. Esto se debe a que el proceso para dar de baja a un empleado requería de un procedimiento interno a la empresa que no se ha podido implementar por continuar con el proyecto de forma autónoma.

## 7.4. Comparativa de las horas estimadas/reales

| PAQUETES DE TRABAJO | | TIEMPO ESTIMADO | TIEMPO REAL |
|---|---|---|---|
| ID | NOMBRE | | |
| **1** | **Gestión del proyecto** | **40 h** | **32 h** |
| 1.1 | Planificación | 15 h | 15 h |
| 1.2 | Seguimiento y control | 15 h | 10 h |
| 1.3 | Reuniones | 10 h | 7 h |
| **2** | **Desarrollo del proyecto** | **270 h** | **297 h** |
| 2.1 | Estudio y análisis de tecnologías | 25 h | 25 h |
| 2.2 | Back-end | 25 h | 22 h |
| *2.2.1* | *Base de datos* | 15 h | 12 h |
| *2.2.2* | *API GraphQL* | 5 h | 7 h |
| *2.2.3* | *Eventos (funciones serveless)* | 5 h | 3 h |
| 2.3 | Front-end | 150 h | 160 h |
| *2.3.1* | *Base de la plataforma* | 20 h | 20 h |
| *2.3.2* | *Gestión de recursos humanos* | 60 h | 90 h |
| *2.3.3* | *Portal de empleados* | 30 h | 25 h |
| *2.3.4* | *Criptomonedas* | 40 h | 25 h |
| 2.4 | Blockchain | 70 h | 90 h |
| **3** | **Memoria y defensa del TFG** | **35 h** | **35 h** |
| 3.1 | Memoria | 25 h | 35 h |
| 3.2 | Presentación | 10 h | - |
| **TOTAL** | | **345 h** | **364 h** |

*Tabla 6. Comparativa horas estimadas/reales*

Como se puede apreciar, se han producido desvíos en la estimación de tiempos planificada. Esta desviación no ha sido muy notoria si solamente tenemos en cuanta las horas totales, pero si nos fijamos en las desviaciones de cada fase vemos que esta desviación no se ha distribuido de manera equitativa.

La fase de gestión del proyecto es la única en la que se ha invertido menos tiempo del estimado. Esto se debe a que el seguimiento y control de proyecto no ha sido tan costoso como se esperaba, y el número de reuniones se redujeron después de seguir con el proyecto de manera independiente.

En la fase de desarrollo del proyecto se ha invertido más tiempo del estimado debido principalmente a la implementación del módulo de gestión de recursos humanos, que requirió de más trabajo por el gran número de funcionalidades con las que cuenta. También requirió más tiempo el desarrollo de la criptomoneda, aunque su implementación en el front-end ha tomado menos tiempo del esperado.

Finalmente, la fase de memoria y defensa del TFG no ha tenido desviaciones, aunque esto se debe a que la preparación de la defensa no se ha realizado a fecha del depósito de la memoria. La realización de la memoria ha exigido un esfuerzo mayor del previsto debido a las revisiones que se han realizado con el tutor académico y a la dificultad subestimada que requiere escribir un documento de este tipo.

# 8. CONCLUSIONES

El desarrollo de este proyecto ha sido más que satisfactorio. He aprendido a crear una aplicación totalmente funcional desde el back-end hasta el front-end incluyendo funcionalidades blockchain a través de la creación de una criptomoneda propia, siendo este último punto el que personalmente me resulta más interesante.

Antes de comenzar con este proyecto mis conocimientos sobre desarrollo de aplicaciones web y blockchain eran muy limitados, por lo que la mayor parte de los conocimientos reflejados en esta memoria han sido adquiridos durante el desarrollo de este proyecto. Tengo que agradecer a Bosonit S.L. y Nter S.L. el haberme permitido experimentar con tecnologías novedosas y con gran futuro en las diferentes áreas de mi proyecto, como son React en el front-end, GraphQL en el back-end y Ethereum en blockchain.

También quiero agradecer a mi tutor académico Eloy Javier Mata Sotés su apoyo, especialmente tras el gran cambio en el alcance y dirección del proyecto que hizo la empresa, permitiéndome seguir con este trabajo de manera independiente.

Este no ha sido el único cambio que ha sufrido el proyecto, ya que durante el tiempo que se estuvo desarrollando este trabajo para la empresa surgieron muchos cambios de menor envergadura. La elección de una metodología iterativo-incremental ha demostrado ser la más adecuada cuando no hay una definición clara del proyecto.

En este proyecto se han empleado un gran número de herramientas y tecnologías. Me gustaría destacar las siguientes lecciones aprendidas y recomendaciones al respecto:

- Recomiendo el uso de Hasura GraphQL engine para la creación de APIs GraphQL, ya que además de crearte la API automáticamente también te permite gestionar los permisos sobre cada tabla de tu base de datos por rol de usuario de manera muy intuitiva.
- Material UI facilita la implementación de un diseño simple y elegante en aplicaciones realizadas con React, aunque tiene el inconveniente de no contar con muchas posibilidades de personalización.
- Las herramientas ofrecidas por la suite de Truffle han sido especialmente útiles, ya que facilitan el desarrollo de smart contracts, su testeo y despliegue.
- Los tutoriales de CryptoZombies son los que más me han ayudado a aprender cómo desarrollar smart contracts. Son muy completos y entretenidos y siguen ampliando su contenido.
- Para encontrar ayuda sobre problemas de desarrollo, además de la ampliamente conocida página web Stack Overflow me ha sido de gran ayuda Medium. Medium cuenta con un gran número de artículos y tutoriales de corta duración para aprender rápidamente.

Aunque finalmente este ha sido un proyecto independiente a la empresa para la que se propuso, la criptomoneda que se ha desarrollado se empleará en una nueva aplicación de la empresa. Además, los conocimientos y experiencia adquiridos son la parte más valiosa de este proyecto.

# 9. BIBLIOGRAFÍA


- Auth0. Documentación oficial. Tutorial de configuración para aplicación React
  https://auth0.com/docs/quickstart/spa/react

- Hasura GraphQL engine. Documentación oficial
  https://hasura.io/docs/1.0/graphql/manual/index.html

- Heroku. Documentación técnica oficial
  https://devcenter.heroku.com/categories/reference

- React. Documentación oficial
  https://reactjs.org/docs/getting-started.html

- Material UI. Página web oficial
  https://material-ui.com/es/

- Apollo Client. Documentación oficial para React
  https://www.apollographql.com/docs/react/

- Truffle Suite:
  - Entorno de desarrollo Truffle. Documentación oficial
    https://www.trufflesuite.com/docs/truffle/overview
  - Blockchain personal con Ganache. Documentación oficial
    https://www.trufflesuite.com/docs/ganache/overview

- Stack Overflow. Foro de preguntas y respuestas para desarrolladores
  https://stackoverflow.com/

- Medium. Servicio de publicación de artículos y tutoriales
  https://medium.com/

- CryptoZombies. Tutoriales interactivos para el desarrollo de smart contracts
  https://cryptozombies.io/